NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MOBILITY MODELING AND ESTIMATION FOR DELAY TOLERANT UNMANNED GROUND VEHICLE NETWORKS

by

Timothy M. Beach

June 2013

Thesis Advisor: Preetha Thulasiraman
Co-Advisor: Grace Clark

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2013	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE MOBILITY MODELING AND ESTIMATION FOR DELAY TOLERANT UNMANNED GROUND VEHICLE NETWORKS		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Timothy M. Beach			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

An ad hoc unmanned ground vehicle (UGV) network operates as an intermittently connected mobile delay tolerant network (DTN). The path planning strategy in a DTN requires mobility estimation of the spatial positions of the nodes as a function of time. The purpose of this thesis is to create a foundational mobility estimation algorithm that can be coupled with a cooperative communication routing algorithm to provide a basis for real time path planning in UGV-DTNs. In this thesis, we use a Gauss-Markov state space model for the node dynamics. The measurements are constant power received signal strength indicator (RSSI) signals transmitted from fixed position base stations. An extended Kalman filter (EKF) is derived for estimating of coordinates in a two-dimensional spatial grid environment. Simulation studies are conducted to test and validate the models and estimation algorithms. We simulate a single mobile node traveling along a trajectory that includes abrupt maneuvers. Estimation performance is measured using zero mean whiteness tests on the innovations sequences, root mean squared error (RMSE) of the state estimates, weighted sum squared residuals (WSSRs) on the innovations, and the posterior Cramer-Rao lower bound (PCRLB). Under these performance indices, we demonstrate that the mobility estimator performs effectively.

14. SUBJECT TERMS Unmanned ground vehicle, delay-tolerant network, mobility estimation, Gauss-Markov model, extended Kalman filter, nonlinear dynamic system, estimation performance indices	**15. NUMBER OF PAGES** 119
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MOBILITY MODELING AND ESTIMATION FOR DELAY TOLERANT UNMANNED GROUND VEHICLE NETWORKS

Timothy M. Beach
Lieutenant, United States Navy
B.S.S.E., United States Naval Academy, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2013

Author: Timothy M. Beach

Approved by: Preetha Thulasiraman
Thesis Advisor

Grace Clark
Thesis Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

An ad hoc unmanned ground vehicle (UGV) network operates as an intermittently connected mobile delay tolerant network (DTN). The path planning strategy in a DTN requires mobility estimation of the spatial positions of the nodes as a function of time. The purpose of this thesis is to create a foundational mobility estimation algorithm that can be coupled with a cooperative communication routing algorithm to provide a basis for real time path planning in UGV-DTNs. In this thesis, we use a Gauss-Markov state space model for the node dynamics. The measurements are constant power received signal strength indicator (RSSI) signals transmitted from fixed position base stations. An extended Kalman filter (EKF) is derived for estimating of coordinates in a two-dimensional spatial grid environment. Simulation studies are conducted to test and validate the models and estimation algorithms. We simulate a single mobile node traveling along a trajectory that includes abrupt maneuvers. Estimation performance is measured using zero mean whiteness tests on the innovations sequences, root mean squared error (RMSE) of the state estimates, weighted sum squared residuals (WSSRs) on the innovations, and the posterior Cramer-Rao lower bound (PCRLB). Under these performance indices, we demonstrate that the mobility estimator performs effectively.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AODV	Ad Hoc on Demand Distance Vector
AR	Autoregressive
ARMA	Autoregressive Moving Average
AUV	Autonomous Unmanned Vehicle
BS	Base Station
CRLB	Cramer-Rao Lower Bound
DSR	Dynamic Source Routing
DTN	Delay Tolerant Network
EKF	Extended Kalman Filter
FIM	Fisher Information Matrix
GPS	Global Positioning System
HMM	Hidden Markov Model
KF	Kalman Filter
MSE	Mean Squared Error
PCRLB	Posterior Cramer-Rao Lower Bound
PF	Particle Filter
QoS	Quality of Service
RBPF	Rao-Blackwellized Particle Filter
RMSE	Root Mean Squared Error
RSSI	Received Signal Strength Indicator
SMC	Sequential Monte Carlo
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
WSSR	Weighted Squared Sum Residual

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

In recent years, there has been increasing interest and engineering activity from academia, industry and governments in the design and deployment of autonomous unmanned vehicles (AUVs). AUVs that are constructed to operate underwater, in air, and on land vary in architecture, capability, and power. In particular, one of the areas that has gained attention is the deployment of unmanned ground vehicles (UGVs). The Navy, Marine Corps and Army have invested monetary resources to the development of UGVs because of their potential to operate in a wide variety of situations [1].

With the introduction of unmanned vehicles, the traditional concept of warfare has shifted to a network centric view of military systems. This involves the integration of communication networking, particularly wireless networking, and information sharing into tactical military operations. This shift towards network centric warfare has led to the need for robust and reliable communications among groups of UGVs.

A UGV network operates as an intermittently connected mobile ad hoc network, otherwise known as a delay tolerant network (DTN) [2]. DTNs have gained considerable attention from the research community as a means of addressing the path planning problem in partitioned networks deployed in environments where infrastructures cannot be installed. Specifically, the problem of routing information between pairs of UGV nodes requires effective path planning protocols to be developed. In order to implement such protocols, it is necessary to have an understanding of the environment in which UGVs are deployed. This is known as situational awareness and includes a real-time understanding of the terrain, activities of other UGVs in the same command, and self-management. A primary factor in obtaining information for situational awareness is the dynamic mobility of each individual UGV node. The dynamic nature of the UGV-DTN requires the path planning protocol to react to the mobility of each individual UGV. Obtaining knowledge about the mobility of the UGVs requires estimation of the position, velocity and acceleration of the UGV-DTN at a given time and is an integral part of the path planning strategy.

In this regard, the overall UGV-DTN system design requires solution of the following two component problems: (1) *Mobility Estimation*: We must develop a set of mobility estimation algorithms that will achieve realistic estimates of the positions of the individual UGV nodes within the DTN, and (2) *Path Planning*: We must develop a path planning strategy using the mobility estimation results as inputs to achieve cooperation among individual UGV nodes for routing.

The research contributions in the networking literature currently take one of two basic approaches to the problem of coupling mobility estimation with path planning (routing) protocols in ad hoc networks: (1) A new mobility estimation algorithm is proposed based upon a constrained spatial grid of cells and Markov-class models (Hidden Markov Models, etc.) [3], [4]. This new mobility estimation algorithm is then coupled with a standard routing protocol such as Ad Hoc On Demand Distance Vector (AODV) or Dynamic Source Routing (DSR), both of which can be found in the NS2 software package widely used for networking simulations [5], [6]. (2) A new routing protocol is proposed, and then it is coupled with standard mobility estimation models like random walk and random waypoint, also found in the NS2 simulation platform [7], [8]. Both [7] and [8] showed that these models impair the accuracy of ad hoc routing algorithms.

Thus, to represent the nodes in a UGV-DTN in a practical setting requires modeling of dynamic movement and several kinds of uncertainty. The node dynamics can best be described by a set of differential equations that include stochastic process noise. The node measurements can be described by algebraic equations that include stochastic measurement noise. Therefore, we chose to use Gauss-Markov state space models which exploit differential equations for the dynamics and an algebraic measurement model [9], [10]. The Gauss-Markov system model forms the basis for model based estimators such as the Kalman Filter (KF). Several research teams have used Kalman type estimators to attack the mobility estimation problem. For example, Zaidi et al. [11] used a simple autoregressive (AR) model as a basis for an Extended Kalman Filter (EKF) in a mobility tracking scheme. Recently, Kalman based filter prediction and multicriteria decision theory have been used in DTNs to choose the next best hop for message delivery [12], [13].

In this regard, the contribution of the research proposed in this thesis lies in the creation of algorithms for both the mobility estimation and the routing protocol that are new to the networking literature. This thesis focuses on the mobility estimation algorithm, while future work is proposed for the development of the network routing protocol. The mobility estimation approach in the thesis exploits a general two-dimensional spatial grid setting, a Gauss-Markov state space dynamic model, a first-order semi-Markov model for the command function, and received signal strength indicator (RSSI) signals for the measurements. The use of signal processing and control techniques for mobility estimation in an ad hoc network is new to the networking literature.

Thus, the aim of this thesis is to provide the foundational algorithm for mobility prediction and estimation such that it can be coupled with a cooperative communication routing algorithm to provide a basis for real time cooperative planning in UGV-DTNs. This thesis makes the following contributions:

- Mobility estimation in ad hoc general spatial grid settings is explored. Existing signal models based on the ad hoc general spatial grid setting and estimation algorithms for both linear and nonlinear models and their uncertainty cases are discussed. Gauss-Markov and semi-Markov type signal models along with the EKF estimation algorithm are chosen for use in the UGV-DTN mobility prediction and estimation algorithm.
- The chosen mobility estimation models are presented. The model for the state of the mobile node and the measurement (observation) model are summarized. The Jacobian matrix required by the EKF is derived.
- The EKF algorithm is developed. The dynamic equations are formulated as an observable continuous-time Gauss-Markov system model. The discrete-time nonlinear Gauss-Markov model and discrete-time EKF algorithm are derived for the UGV-DTN. Performance measures for EKF evaluation and tuning are presented.

- The UGV-DTN mobility prediction and estimation algorithm is simulated in MATLAB. The performance of the EKF is evaluated and discussed.

The EKF algorithm operates recursively in time, meaning that the current state vector estimate is a function of only the estimate at the last time step. The storage of additional past information is not required, so storage resource utilization for individual UGV nodes is minimized.

In our performance evaluations, we simulate a single node traveling along a trajectory that includes abrupt maneuvers. Estimation performance is assessed with zero mean whiteness tests on the innovation sequences, root mean squared error (RMSE) of the state estimates, weighted squared sum residuals (WSSRs) on the innovations, and the posterior Cramer-Rao lower bound (PCRLB). The algorithm is shown to implement efficient mobility tracking of UGV nodes in a wireless network. We demonstrate that the mobility estimator performs effectively and therefore can be legitimately integrated into new cooperative routing protocol with enhanced accuracy.

References

[1] R. O'Rourke, *Unmanned Vehicles for U.S. Naval Forces: Background and Issues for Congress*, CRS Report for Congress, October 2006. Available: http:www.fas.org/sgp/crs/weapons/RS21294.pdf.

[2] E. Kuiper and S. Nadjm-Tehrani, "Geographical Routing with Location Service in Intermittently Connected MANETs," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 2, pp. 592-604, February 2011.

[3] P.S. Prasad and P. Agarwal, "Effect of Mobility Prediction on Resource Utilization in Wireless Networks," *Proc. IEEE WCNC*, pp. 1–6, 2010.

[4] P.S. Prasad and P. Agarwal, "A Generic Framework for Mobility Prediction and Resource Utilization in Wireless Networks," *Proc. IEEE COMSNETS*, pp. 1–10, 2010.

[5] P.S. Prasad, P. Agarwal, and K.M. Sivalingam, "Effects of Mobility in Hierarchical Mobile Ad Hoc Networks," *Proc. IEEE CCNC*, pp. 1–5, 2009.

[6] A. Jardosh, E.M. Belding, K.C. Almeroth, and S. Suri, “Towards Realistic Mobility Models for Mobile Ad Hoc Networks,” *Proc. of ACM Mobicom*, pp. 217-229, 2003.

[7] C. Bettstetter, G. Resta and P. Santi, “The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks,” *IEEE Transactions on Mobile Computing*, vol. 2, no. 3, pp. 257-269, September 2003.

[8] G. Lin, G. Noubir and R. Rajaraman, “Mobility Models for Ad Hoc Network Simulation,” *Proc. IEEE INFOCOM*, pp. 1–10, 2004.

[9] A. Vahdat and D. Becker, “Epidemic Routing for Partially Connected Ad Hoc Networks,” *Tech. Report CS-2000-06*, Department of Computer Science, Duke University, April 2000, Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.6151&rep=rep1&type=pdf.

[10] S. Jain, K. Fall, and R. Patra, “Routing in Delay Tolerant Networks,” *Proc. ACM Special Interest Group on Data Communications (SIGCOMM)*, pp. 145–158, 2004.

[11] Z.R. Zaidi and B.L. Mark, “Mobility Tracking Based on Autoregressive Models,” *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 32–43, January 2011.

[12] E. Amar and S. Boumerdassi, “A Scalable Mobility-Adaptive Location Service with Kalman Based Prediction,” *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 593–598, 2011.

[13] M. Musolesi and C. Mascolo, “CAR: Context Aware Adaptive Routing for Delay Tolerant Mobile Networks,” *IEEE Wireless Transactions on Mobile Computing*, vol. 8, no. 2, pp. 246–260, February 2009.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife. I could not have completed my graduate research without her love, commitment, support and inspiration through all of the long days and late nights. Her understanding and patience knows no bounds. I would also like to thank my family for their support and inspiration throughout my life.

I cannot thank my co-advisors Dr. Preetha Thulasiraman and Dr. Grace Clark enough. Their unparalleled patience, kindness, encouragement, and technical prowess and endless support, wisdom and motivation enriched every stage of my graduate work. The mentorship and guidance were vital to my understanding of the topic and quality of the thesis. My graduate research would not have been a success without them. I would also like to thank Dr. Gary Hutchins, Donna Miller, and Sam Barone for their kindness, support, and contribution to the completion of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION AND MOTIVATION

In recent years, there has been increasing interest and engineering activity from academia, industry and governments in the design and deployment of autonomous unmanned vehicles (AUVs). AUVs that are constructed to operate underwater, in air, and on land vary in architecture, capability, and power. In particular, one of the areas that has gained attention is the deployment of unmanned ground vehicles (UGVs). The Navy, Marine Corps and Army have invested monetary resources to the development of UGVs because of their potential to operate in a wide variety of situations [1].

With the introduction of unmanned vehicles, the traditional concept of warfare has shifted to a network centric view of military systems. This involves the integration of communication networking, particularly wireless networking, and information sharing into tactical military operations. This shift towards network centric warfare has led to the need for robust and reliable communications among groups of UGVs.

A UGV network operates as an intermittently connected mobile ad hoc network, otherwise known as a delay tolerant network (DTN) [2]. DTNs have gained considerable attention from the research community as a means of addressing the path planning problem in partitioned networks deployed in environments where infrastructures cannot be installed. The ability to implement effective communication protocols among UGVs depends very much on the strategy of understanding the environment in which they are deployed. This is known as situational awareness and includes a real-time understanding of the terrain, activities of other UGVs in the same command, and self-management. A primary factor in obtaining information for situational awareness is the dynamic mobility of each individual UGV. The dynamic nature of the UGV-DTN requires path planning protocol react to the mobility of each individual UGV. Obtaining knowledge about the mobility of the UGVs requires estimation of the position, velocity and acceleration of the UGV-DTN at a given time and is an integral part of the path planning strategy.

In this regard, the overall UGV-DTN system design requires solution of the following two component problems:

- *Mobility Estimation*: We must develop a set of mobility estimation algorithms that will achieve realistic estimates of the positions of the individual UGV nodes within the DTN.
- *Path Planning*: We must develop a path planning strategy using the mobility estimation results as inputs to achieve cooperation among individual UGV nodes for routing.

This thesis focuses on the mobility estimation problem. The development of the path planning algorithm and the integration of the two components is left for future research.

A. THE UGV-DTN SYSTEM MODEL

The network environment considered in this thesis is shown in Figure 1. Spatially distributed UGV cluster islands are assumed to be connected via an unmanned aerial vehicle which can act as a relay node to carry information among groups of UGV cluster islands. UGVs within a cluster island communicate cooperatively to forward messages from source to destination within one cluster island. Geo-location using the Global Positioning System (GPS) is assumed to be available on an unmanned aerial vehicle (UAV) and on a sub-set of UGV nodes called anchor nodes, which carry GPS in addition to their built-in received signal strength indicator (RSSI) sensors. As a starting point, the cooperative communication protocols required within one individual UGV cluster island must first be explored. The proposed overall signal/data flow for a single cluster island is shown in Figure 2. All UGV nodes can communicate with other nodes in their cluster island. However, there is no direct communication among nodes in different clusters. The UAV and all of the UGV nodes contain sensors that produce measurements as indicated in Figure 2. Each UGV node carries a set of sensors, as does the UAV. For this study, the set of possible sensor types includes both RSSI and GPS sensors. The mobile nodes and the UAV pass measurement signals/data to the routing algorithms and the mobility predication algorithms. As shown in Figure 2, the purpose of the mobility prediction algorithms is to produce predictions of position vs. time (and sometimes velocity vs. time) for a particular node or nodes in the cluster island.

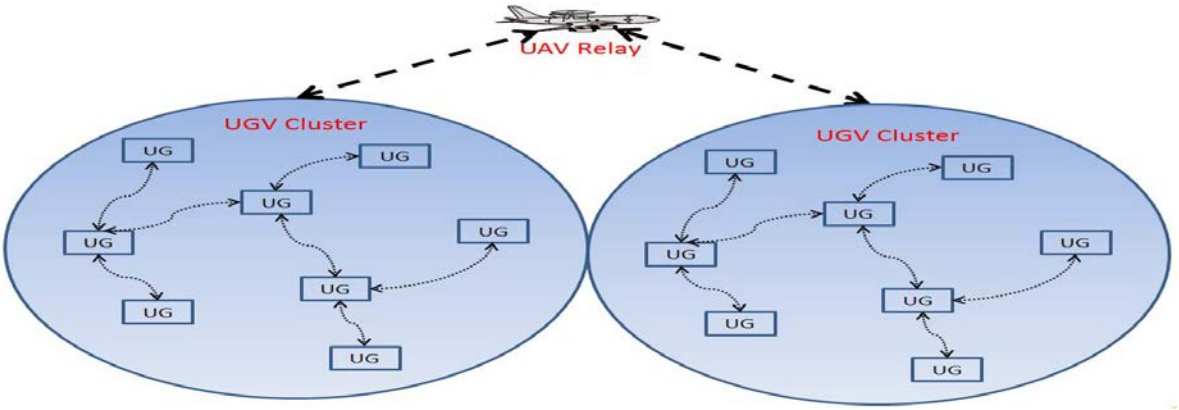

Figure 1. A clustered UGV-DTN communicating wirelessly within each cluster and to and from a UAV.

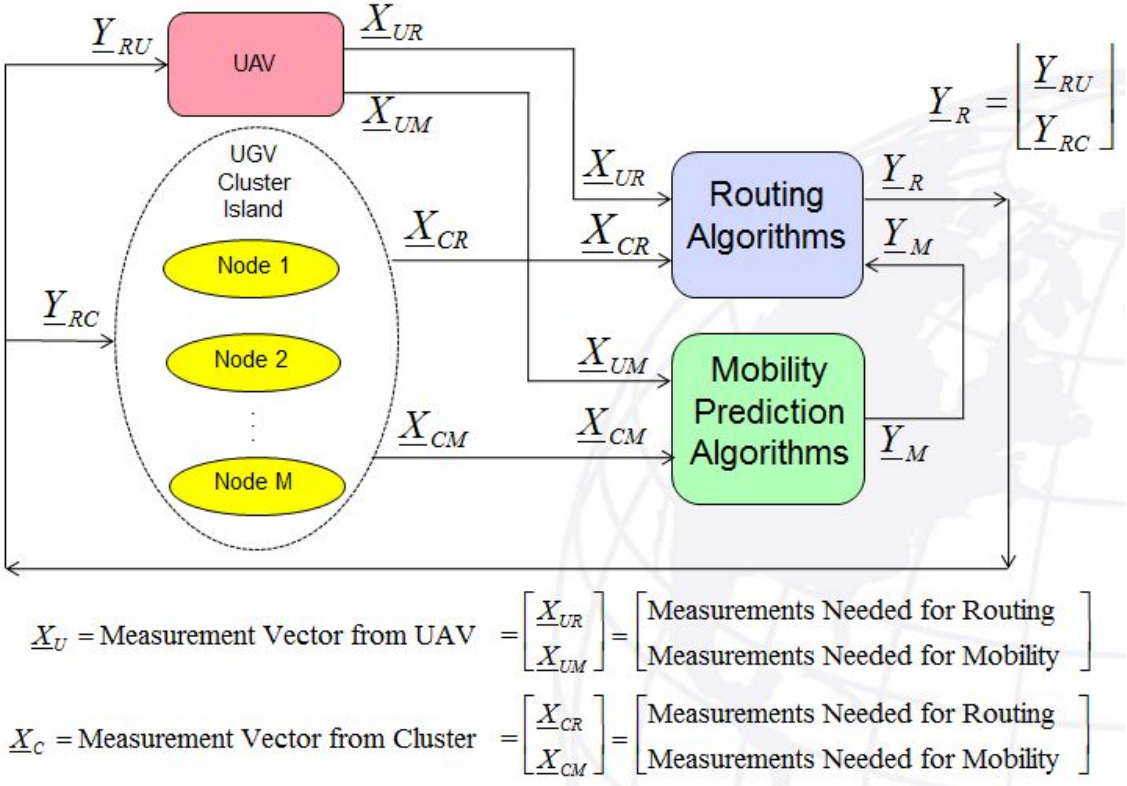

Figure 2. Block diagram of the overall signal/data flow for a UGV-DTN.

To further breakdown the overall problem, the measurements obtained from one UGV node will be used to construct the measurement vector for the mobility estimation and routing protocol as shown in Figure 3.

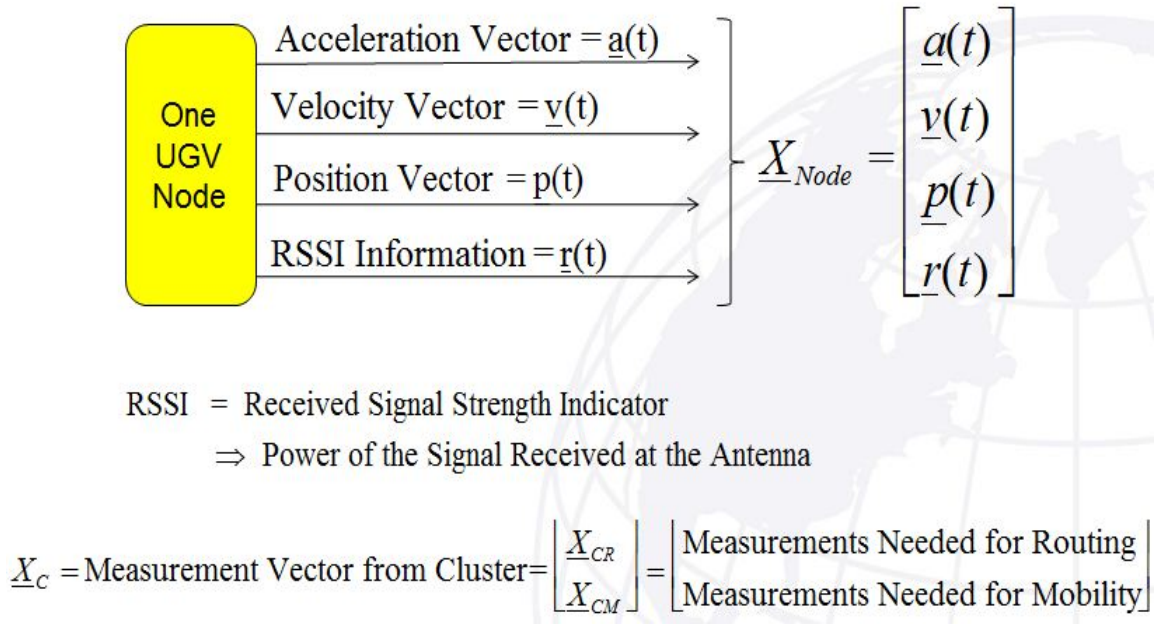

Figure 3. Block diagram of the overall signal/data flow for one UGV-DTN node in a cluster.

B. COMMUNICATIONS PARADIGM OF THE UGV-DTN

Reliability is an essential feature of communications in wireless networks and is generally coupled with design in path planning and routing. Reliability in UGV-DTNs is a quality of service (QoS) issue that depends on resource availability (capacity and bandwidth) and the topology of the network at an instant of time [3].

There are many works focused on joint routing and mobility prediction in dynamic wireless DTNs that provide a strong foundation for research. DTNs have traditionally been modeled as mobile ad hoc networks that have intermittent connectivity [4]. In such networks, the notion of combinatorial stability is introduced as a way of determining loop free paths in a mobile setting. The ability to communicate is proven to degrade with increasing mobility and inconsistent topology information. Given

a time of duration *t*, an ad hoc network is combinatorially stable if and only if topology changes occur sufficiently slowly enough to allow successful propagation of all topology updates [5]. The assumption is that the network remains quasi-static for a period of time during which route updates can occur.

The notion of combinatorial stability facilitates research simulations and is critical for QoS assurance in ad hoc networks. In generic DTNs and in this case, UGV-DTNs, it is not possible to assume a quasi-static nature. UGV-DTNs are to be versatile and utilized in the operational environment where nothing can be guaranteed and communication connectivity may be denied for some time due to intentional jamming or simply excessive mobility. Thus, the nature of the UGV-DTN deployment indicates that combinatorial stability is a stringent and sometimes impractical assumption. QoS in UGV-DTNs depends on the integration of mobility and situational awareness into path planning algorithms to maximize the probability of connectivity between UGV pairs and minimize aggregate resource consumption [6], [7].

DTN routing protocols have been improved upon through the use of mobility prediction. When prediction is used in DTN routing, the mobility model is an important factor. Modeling the nodes in a UGV-DTN requires modeling dynamic movement and several kinds of uncertainty. The node dynamics can best be described by a set of differential equations that include stochastic process noise. The node measurements can be described by algebraic equations that include stochastic measurement noise. Therefore, we chose to use Gauss-Markov state space models which exploit differential equations for the dynamics and an algebraic measurement model [8], [9]. The Gauss-Markov system model forms the basis for model based estimators such as the Kalman Filter (KF). Several research teams have used Kalman type estimators to attack the mobility estimation problem. For example, Zaidi et al. [10] used a simple autoregressive (AR) model as a basis for an Extended Kalman Filter (EKF) in a mobility tracking scheme. Recently, Kalman based filter prediction and multicriteria decision theory have been used in DTNs to choose the next best hop for message delivery [11], [12].

C. APPROACHES TO STOCHASTIC MOBILITY ESTIMATION

The mobility estimation literature for two-dimensional spatial planes generally deals with two fundamental types of physical two-dimensional spatial settings: (1) a known, predefined constrained grid of spatial cells (not ad hoc) and (2) a general spatial grid that uses a reference frame defined by geospatial coordinates (ad hoc). These coordinates can be estimated using measured signals from known base stations (BSs), the GPS, etc.

1. Setting: Constrained Grid of Spatial Cells

Perhaps the largest portion of the mobility estimation literature deals with settings in which the movements of mobile nodes (users) are constrained on a known, predefined grid of spatial cells. Such a setting does not define an ad hoc network. For example, in [13] the spatial grid is defined on the grounds of a university campus, and contains "landmarks" such as buildings. Settings of this type are excellent for solving a useful class of real-world problems (i.e., Wireless Local Area Networks in a known constrained spatial area) as they are able to exploit prior knowledge very efficiently [14]. In such settings, the measurements arise in the form of user movement sequences that are collected by a centralized wireless access point controller and stored for use by the mobility estimation algorithms. The signal model for such settings is commonly a form of Markov or Semi-Markov model [11], [14], Hidden Markov Model (HMM) or variants of these models. These models are very well suited to settings that use a grid of cells. The performance indices for such settings commonly consist of probabilities and indices such as likelihood of next cell transition, likelihood of a user being in a particular state after N transitions, probability density function of future contact times and expected spatial-temporal traffic load at each location in a network's coverage area. The estimation algorithms used are derived specifically for the Markov signal models, (i.e., the Viterbi algorithm and the Baum-Welch algorithms for HMMs) [13], [14], [15].

2. Setting: General Spatial Grid

The ad hoc General Spatial Grid Setting is generally most appropriate for mobile ad hoc networks based on a DTN architecture, particularly in military applications. In

operational settings, it is not likely that one can easily define a constrained grid of spatial cells on which to operate. One must operate wherever one is deployed. The measurements in this setting usually involve RSSI signals from known base stations [8], [13], [15]. The signal model is usually some form of Gauss-Markov (state space differential equations and measurement equations). These can include AR moving average (ARMA) models and their many variants [12], [16], [17]. These models are sometimes augmented with a semi-Markov model to represent uncertain accelerations, etc. [19], [20]. Models of this type efficiently accommodate a wide variety of measurement types, including RSSI, time-of-arrival, angle-of-arrival, and/or GPS measurements. The key requirement is that the model must be observable in the estimation and control theory sense [19], [20]. This means the available information must be sufficient to allow the estimation of the system states. The performance indices generally include mean squared error (MSE), the Cramer-Rao lower bound (CRLB), and the posterior CRLB (PCLRB) [12], [16], [20]. The estimation algorithms usually consist of a Kalman Filter (KF), EKF, Unscented KF or particle filter (PF), also called a sequential Monte Carlo (SMC) filter, depending on whether or not the system model is linear or nonlinear and the uncertainties (noise processes) are Gaussian or Non-Gaussian. In general, the KF is appropriate for the linear, Gaussian case. The EKF and the Unscented KF can sometimes be used in the nonlinear, Gaussian case, and the PF is used in the nonlinear, non-Gaussian case, when other algorithms do not perform sufficiently well [12], [21]. One variation is the Rao-Blackwellized particle filter (RBPF), which uses the KF for the linear part of the processing and the PF for the nonlinear part [14], [21]. Some published algorithms assume that the model parameters are known a priori [14]. This requires good first principles modeling and/or system identification/calibration step prior to using the algorithm. At least one research team has proposed an algorithm that jointly estimates the model parameters and the systems states simultaneously [8]. This can have performance advantages if the model is simple enough to allow on-line parameter estimation.

D. MOTIVATION AND CONTRIBUTIONS OF THE THESIS

The contributions in the networking literature discussed in Sections IB and IC currently take one of two basic approaches to the problem of coupling mobility estimation with routing protocols in ad hoc networks: (1) A new mobility estimation algorithm is proposed based upon a constrained spatial grid of cells and Markov-class models (Hidden Markov Models, etc.) [22], [23]. This new mobility estimation algorithm is then coupled with a standard routing protocol such as Ad Hoc On Demand Distance Vector (AODV) or Dynamic Source Routing (DSR), both of which can be found in the NS2 software package widely used for networking simulations [24] ,[25]. (2) A new routing protocol is proposed, and then it is coupled with standard mobility estimation models like random walk and random waypoint, also found in the NS2 simulation platform [26], [27]. Both [26] and [27] showed that these models impair the accuracy of ad hoc routing algorithms.

The contribution of the research proposed in this thesis lies in the creation of algorithms for both the mobility estimation and the routing protocol that are new to the networking literature. This thesis focuses on the mobility estimation algorithm, while future work is proposed for the development of the network routing protocol. The mobility estimation approach in the thesis exploits a general two-dimensional spatial grid setting, a Gauss-Markov state space dynamic model, and a first-order semi-Markov model for the command function. The use of signal processing and control techniques for mobility estimation in an ad hoc network is new to the networking literature.

Thus, the aim of this thesis is to provide the foundational algorithm for mobility prediction and estimation such that it can be coupled with a cooperative communication routing algorithm to provide a basis for real time cooperative planning in UGV-DTNs. This thesis makes the following contributions:

- Stochastic mobility prediction in ad hoc general spatial grid settings is explored. Existing signal models based on the ad hoc general spatial grid setting and estimation algorithms for both linear and nonlinear models and their uncertainty cases are discussed. Gauss-Markov and semi-Markov type signal models along with the EKF estimation algorithm are chosen for use in the UGV-DTN mobility prediction and estimation algorithm.

- The chosen mobility estimation models are presented. The model for the state of the mobile node and the measurement (observation) model are summarized. The Jacobian matrix required by the EKF is derived.
- The EKF algorithm is developed. The dynamic equations are formulated as an observable continuous-time Gauss-Markov system model. The discrete-time nonlinear Gauss-Markov model and discrete-time EKF algorithm are derived for the UGV-DTN. Performance measures for EKF evaluation and tuning are presented.
- The UGV-DTN mobility prediction and estimation algorithm is simulated in MATLAB. The performance of the EKF is evaluated and discussed.

E. ORGANIZATION OF THE THESIS

The remainder of the thesis is organized as follows. Stochastic mobility estimation in the ad hoc general spatial grid setting is discussed in Chapter II. The signal mobility estimation models used to derive the Jacobian for use in the EKF is discussed in Chapter III. The EKF algorithm and performance measures are developed in Chapter IV. The performance evaluation and results of the EKF simulation experiment, as well as the process of EKF tuning, are presented in Chapter V. The conclusions and a discussion of directions for future work are provided in Chapter VI. The Appendix contains the MATLAB m-files and functions used in this research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. STOCHASTIC MOBILITY PREDICTION IN THE GENERAL SPATIAL GRID SETTING

In this chapter we focus on a subset of literature that is most appropriate for our problem. While Chapter I provided a general literature review of the research area, the following sections provide a review of [13], [15], and [18] studied for adaptation in the UGV-DTN estimation scenario. The military operational setting for mobile ad hoc networks based on a DTN architecture makes use of the ad hoc general spatial grid setting. This setting is most appropriate for mobile ad hoc networks based on a DTN architecture. The purpose of the mobility algorithm in the UGV-DTN is to produce estimates of position over time, and sometimes velocity and acceleration over time, within the general spatial grid setting. The algorithm design requires several key problem specifications, or attributes. The key attributes are as follows: (1) the operational mission setting and physical constraints, (2) the set of available sensor measurements or observations, (3) an appropriate physics model, (4) an appropriate performance index or set of performance indices, and (5) an appropriate estimation/tracking algorithm or set of algorithms. We describe and compose algorithms in terms of the five key attributes.

A. MOBILITY TRACKING IN WIRELESS AD HOC NETWORKS

Zaidi et al. [15] studies ad hoc networks with intermittent connectivity. User mobility makes the topology of an ad hoc network dynamic over time complicating the routing and flow of information. The algorithm for mobility tracking developed in [15] uses RSSI measurements from neighboring nodes modeled as a linear system driven by a discrete semi-Markov process in combination with an efficient averaging filter and an EKF. A scheme for a local coordinate system for ad hoc networks using relative distances between nodes is recommended [15].

The proposed algorithm allows robust mobility tracking in ad hoc networks using RSSI measurements. Estimated parameters are used to determine the autocorrelation function. The discrete-time processes for the three mobile nodes are modeled as independent semi-Markov processes. The mobility tracking algorithm pre-filters the

observations prior to applying an EKF for mobility state estimation. Root MSE (RMSE) is used as the performance measure. The algorithm is able to follow mobile trajectories accurately over a wide range of parameter values and provides the following unique advantages over previous proposed algorithms. It requires information about just one stationary network node as opposed to knowing three points to determine the position of a mobile node. The advantage of this is due to the ability of the KF to reduce the observation error of all the three nodes simultaneously. The prior information needed is thus greatly reduced. The proposed mobility tracking algorithm can be applied in a variety of scenarios, such as adaptive clustering, routing, and mobility management in ad hoc networks [15].

B. MODIFIED EKF AND SEQUENTIAL MONTE CARLO FILTER

Yang et al. [18] consider a SMC method for joint mobility tracking and cellular handoff in wireless communication networks. The mobility tracking is based on the measurement of RSSI signals from known base stations. The system dynamics are described by a nonlinear state space model. The movement of the individual node is modeled as a semi-Markov chain with a first-order AR model adopted for random acceleration correlation. The mobility tracking includes estimation of the position and velocity of the mobile node. The EKF is identified as the main technique for solving online estimation in a nonlinear dynamic system. Yang et al. were attempting to solve two problems: (1) online mobility estimation and (2) online prediction of the RSSI at some future time instance. The optimal solutions to both problems were prohibitively complex due to system nonlinearities. Therefore, an SMC estimator is built on the techniques of importance sampling and resampling and provides an online posterior distribution of a node's location and velocity based on a nonlinear state space model. Under the SMC framework, both problems can be solved naturally in a joint fashion. The SMC was compared with the modified EKF and was shown to improve tracking accuracy and minimize the tradeoff between QoS and resource utilization [18]. However, the SMC-based approach comes with a significantly high computational cost.

C. EXTENDED KALMAN FILTER, PARTICLE FILTER AND RAO-BLACKWELLIZED PARTICLE FILTER

Mihaylova et al. [13] also considers a SMC technique for mobility tracking in wireless communication networks by means of RSSI. The technique allows for accurate estimation of mobile position and speed. The command process is represented by a first-order semi-Markov model, which takes values from a finite set of acceleration levels that cover the range of probable acceleration. A PF and RBPF are proposed and analyzed over real and simulated data. A comparison with an EKF is performed with respect to accuracy and computational complexity. With a small number of particles the RBPF gives more accurate results than the PF or the EKF. A PCRLB is calculated and it is compared to the filter's RMSE performance [13]. The designed filters are compared to the EKF technique to identify enhanced performance with respect to scenarios with abrupt maneuvers. Advantages of the RBPF compared with the PF are decreased computational complexity exhibiting similar accuracy with smaller number of particles and smaller peak-dynamic errors during abrupt maneuvers, which is important for practical purposes [13]. It is important to note that without abrupt changes, the EKF performs admirably. In this thesis, we adapt the state space mobility model from [13].

THIS PAGE INTENTIONALLY LEFT BLANK

III. MOBILITY ESTIMATION MODELS

For our analysis we chose to use a Gauss-Markov state space model [15]. The specific model chosen for the implementation of the mobility prediction of a UGV-DTN is a discrete-time variant of the Singer model originally proposed in [28]. Mihaylova et al. [16] has shown that the modified Springer model used by Yang and Wang [20] performs well, is simple, and allows efficient computation of performance indices. This is a Gauss-Markov type model modified to include a discrete semi-Markov type model. The dynamic model for the mobile node is linear, but the measurement model is highly nonlinear.

A. MODEL FOR THE STATE OF THE MOBILE NODE

Let the two-dimensional spatial coordinates be denoted by (x, y). Let k denote the discrete time index, and let T denote the temporal sampling period. We let (x_k, y_k) denote the position of the mobile node at discrete time k. We then denote the speed by $(\dot{x}_k, \dot{y}_k)$ and the acceleration by $(\ddot{x}_k, \ddot{y}_k)$. The parameter α depends on the duration of a maneuver, and is the reciprocal of the maneuver time constant. The state of the mobile node at discrete time k is then denoted by $\underline{x}_k = \left[x_k, \dot{x}_k, \ddot{x}_k, y_k, \dot{y}_k, \ddot{y}_k\right]^T$ where the superscript T denotes vector transpose. The linear state for the mobile node is given by:

$$\underline{x}_k = A(T, \alpha)\,\underline{x}_{k-1} + B_u(T)\,\underline{u}_k + B_w(T)\,\underline{w}_k \tag{3.1}$$

where $\underline{u}_k = \left[u_{x,k}, u_{y,k}\right]^T$ is the discrete-time command process, or system input, and $\underline{w}_k = \left[w_{x,k}, w_{y,k}\right]^T$ is a white Gaussian noise sequence with zero mean and a covariance matrix $Q = \sigma_w^2 I$ where I denotes the unit, or identity matrix. Note that the matrix $A(T, \alpha)$ is a function only of the sampling period and the reciprocal of the maneuver constant, and the matrices $B_u(T)$ and $B_w(T)$ are functions only of the sampling period [16].

Over time, in the real world, a mobile node is likely to have both discontinuous motion and continuous motion. A mobile node is likely to have sudden and unexpected acceleration changes. These could be caused by traffic lights, turns in the road, the need for collision avoidance, etc. Simultaneously, we must account for the fact that node acceleration is likely to be correlated over time, due to momentum. For example, if a node is accelerating at time sample k, then it likely will be accelerating at time sample $k+1$. For these reasons, we model the mobile node as a dynamic system driven by a semi-Markov acceleration process $\underline{a}_k = \underline{u}_k + \underline{r}_k$ as shown in Figure 4. This acceleration is the sum of a two-dimensional semi-Markov driving command $\underline{u}_k = \left[u_{x,k}, u_{y,k}\right]^T$ and a two-dimensional time-correlated random acceleration vector $\underline{r}_k = \left[r_{x,k}, r_{y,k}\right]^T$. The two commands $u_{x,k}$ and $u_{y,k}$ are independent semi-Markov processes acting in the x and y directions [15], [19], [28].

The command $\underline{u}_k$ creates discrete unexpected changes in acceleration, which are modeled as a semi-Markov process with a finite number of states $S_1, S_2, \ldots, S_m$ as shown in Figure 4. A semi-Markov process assumes that we have the Markov state transition probability and random duration of time in one state before it switches to another state [19]. These finite states represent discrete levels of acceleration, which we denote as follows: $M = M_x \times M_y = \{m_1, \ldots, m_M\}$, where M_x and M_y are acceleration levels in the x and y directions in two-dimensional space, and represent states with associated state transition probabilities $\pi_{i,j} = P\left(u_k = m_j \mid u_{k-1} = m_i\right)$, where $i, j = 1, \ldots, M$ and the initial probability distribution $\mu_{i,0} = P\left(m = m_i\right)$ for all $m_i \in \mathrm{M}$ such that $\mu_{i,0} \geq 0$ and $\sum_{i=1}^{M} \mu_{i,0} = 1$ [14].

The random acceleration $\underline{r}_k = \left[r_{x,k}, r_{y,k}\right]^T$ is modeled as a correlated zero mean random vector with a variance designed to cover the "gap" between adjacent acceleration states $S_1, S_2, ..., S_m$. The conditional probability densities of $\underline{a}_k$ given the states are

depicted in Figure 4 for the one-dimensional case in which the acceleration is a scalar. Note that vectors are denoted by bold lettering in this figure [19]. The finite acceleration states in Figure 4 lie in the range $[-A_{\max}, A_{\max}]$.

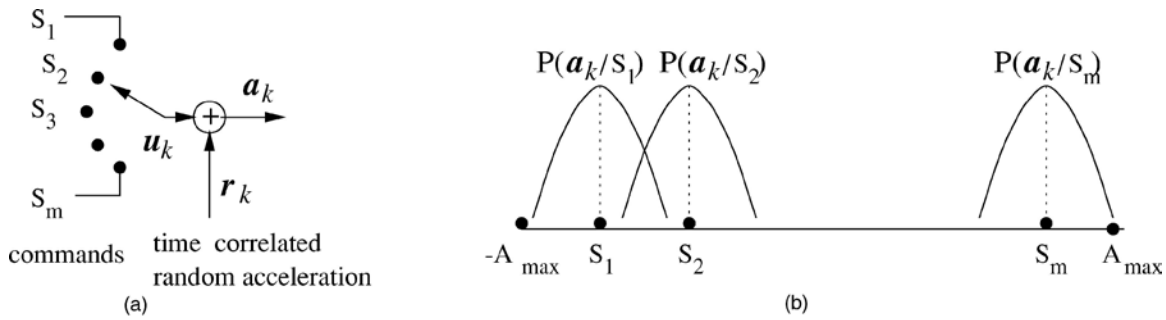

Figure 4. (a) Semi-Markov command acceleration input signal process for the UGV $\underline{a}_k = \underline{u}_k + \underline{r}_k$. (b) Conditional probability densities of $\underline{a}_k$ given the states $S_1, S_2, ..., S_m$ for the 1-D (scalar) case [19].

Let us now specify how the model produces the correlated random accelerations. We can obtain a correlated stochastic process by passing a zero mean white Gaussian process $\underline{w}_k$ through a shaping filter. A commonly used representative model of the autocorrelation function is given by [19]:

$$R_{rr}(\tau) = E\{\underline{r}(t)\underline{r}^T(t+\tau)\} = \sigma_m^2 e^{-\alpha|\tau|} I, \tag{3.2}$$

where $\alpha \geq 0$ and σ_m^2 is the variance of the random acceleration in a single dimension, and α is the reciprocal of the random acceleration time constant. The desired stochastic process can be obtained by passing a zero mean white Gaussian process $\underline{w}_k \sim N[0, R_w^2]$ with covariance $R_w^2 = 2\alpha\sigma_m^2\delta(\tau)I$ through a one-pole AR shaping filter specified by the following difference equation:

$$\underline{r}_{k+1} = -\alpha\underline{r}_k + \underline{w}_k. \tag{3.3}$$

We can combine the modified dynamic state vector $\underline{x}_k = [x_k, \dot{x}_k, y_k, \dot{y}_k]^T$ with $\underline{u}_k$ and $\underline{r}_k$ to obtain:

$$\underline{x}_{k+1} = F\underline{x}_k + E\underline{u}_k + G\underline{r}_k, \tag{3.4}$$

where

$$F = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, G = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{3.5}$$

and $\underline{r}_k$ is correlated in time.

By augmenting the state vector with $\underline{r}_k$, the discrete time dynamic state equation can be expressed in terms of discrete white Gaussian noise $\underline{w}_k$ and the driving command $\underline{u}_k$:

$$\underline{x}_{k+1} = A\underline{x}_k + B\underline{u}_k + \underline{w}_k, \tag{3.6}$$

where $\underline{x}_k = \left[x_k, \dot{x}_k, r_{x,k}, y_k, \dot{y}_k, r_{y,k} \right]^T \triangleq \left[x_k, \dot{x}_k, \ddot{x}_k, y_k, \dot{y}_k, \ddot{y}_k \right]^T$. The node acceleration is extended to include the single-pole filter [21]. The final mobile node dynamic model is summarized as follows:

$$\underline{x}_k = A(T,\alpha)\underline{x}_{k-1} + B_u(T)\underline{u}_k + B_w(T)\underline{w}_k =$$

$$\begin{bmatrix} x_k \\ \dot{x}_k \\ \ddot{x}_k \\ y_k \\ \dot{y}_k \\ \ddot{y}_k \end{bmatrix} = \begin{pmatrix} 1 & T & \frac{T^2}{2} & 0 & 0 & 0 \\ 0 & 1 & T & 0 & 0 & 0 \\ 0 & 0 & \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & \frac{T^2}{2} \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & \alpha \end{pmatrix} \begin{bmatrix} x_{k-1} \\ \dot{x}_{k-1} \\ \ddot{x}_{k-1} \\ y_{k-1} \\ \dot{y}_{k-1} \\ \ddot{y}_{k-1} \end{bmatrix} + \begin{pmatrix} \frac{T^2}{2} & 0 \\ T & 0 \\ 0 & 0 \\ 0 & \frac{T^2}{2} \\ 0 & T \\ 0 & 0 \end{pmatrix} \begin{bmatrix} u_{x,k} \\ u_{y,k} \end{bmatrix} + \begin{pmatrix} \frac{T^2}{2} & 0 \\ T & 0 \\ 1 & 0 \\ 0 & \frac{T^2}{2} \\ 0 & T \\ 0 & 1 \end{pmatrix} \begin{bmatrix} w_{x,k} \\ w_{y,k} \end{bmatrix}. \tag{3.7}$$

B. MEASUREMENT (OBSERVATION) MODEL

The measurements consist of RSSI signals from known-location BSs. Locating a node in a two-dimensional spatial plane requires a minimum of three BSs. Increasing the

number of BSs to seven will improve accuracy [16]. Let M_{BS} denote the number of BSs. We are given measurements of the location $\left(a_{i,k}, b_{i,k}\right)$ of each of the BSs at discrete time k, where $i = 1, \ldots, M_{BS}$. Let us denote the measurement model by a nonlinear vector equation of the form:

$$\underline{z}_k = \underline{h}\left[\underline{x}_k\right] + \underline{\nu}_k \tag{3.8}$$

where $\underline{z}_k$ denotes the measurement vector, $\underline{h}\left[\underline{x}_k\right]$ is an nonlinear function, and $\underline{\nu}_k$ is the measurement noise. The RSSI signal can be modeled as a sum of two terms: path loss $\underline{h}\left[\underline{x}_k\right]$ and shadow fading $\underline{\nu}_k$. The one-pole AR filter in Eq. (3.3) has the effect of attenuating any Rayleigh or Rican Fading. The RSSI signal, measured in decibels (dB), is a signal that a mobile unit receives from a particular base station or anchor node. The RSSI signal of a single BS is modeled by:

$$\underline{z}_{k,i} = z_{0,i} - 10\eta \log_{10}\left(d_{k,i}\left[\underline{x}_k\right]\right) + \nu_{k,i} \tag{3.9}$$

where $\underline{z}_{0,i}$ is a constant characterizing the transmission power of the base station. It is a function of wavelength, antenna height, and gain of node i [16]. The constant η is called the slope index, and it takes on various values, depending on the characteristics of the physical environment (i.e., typically $\eta = 2$ for highways and $\eta = 4$ for microcells in a city). The distance $d_{k,i}\left[\underline{x}_k\right] = \sqrt{\left(x_k - a_{i,k}\right)^2 + \left(y_k - b_{i,k}\right)^2}$ is the distance between the mobile node and the base station i at discrete time k . The process $\underline{\nu}_k = \left[\nu_{k,1}, \ldots, \nu_{k,M_{BS}}\right]$ is the shadowing component. It has been shown to be stationary and uncorrelated both in time and space with white Gaussian distribution $\underline{\nu}_{k,i} \sim N\left[0, \sigma_\nu^2\right]$ for $i = 1, \ldots, M_{BS}$ [16]. The shadowing component can considerably degrade the estimation process, but this difficulty can be overcome by prefiltering in order to reduce observation noise [17].

C. DERIVATION OF THE JACOBIAN MATRIX REQUIRED BY THE EKF

The EKF used for the estimation algorithm requires a Jacobian, or gradient, matrix for approximate linearization of our non-linear measurement. First, we must define the highly nonlinear measurement function $c[\underline{x}_k]$ in the general Gauss-Markov model for our application [14]. By inspection, we see that

$$c[\underline{x}_k] \triangleq h(\underline{x}_k) = z_{0,i} - 10\eta \log_{10}\left(d_{k,i}[\underline{x}_k]\right). \tag{3.10}$$

Let us gather some general relationships and definitions we need for the derivation. First, the general definition of the Jacobian matrix for an EKF is given by [15]:

$$\tilde{\mathbb{C}} \triangleq \left.\frac{\partial \underline{c}[\underline{x}_k]}{\partial \underline{x}_k}\right|_{\underline{x}_k = \hat{\underline{x}}_{k|k-1}}. \tag{3.11}$$

Second, the Euclidean norm of the difference between vectors $\underline{x}$ and $\underline{\theta}$ is given by [14]

$$d_{x,\theta}^2 = \left\|\underline{x} - \underline{\theta}\right\|^2 = \sum\nolimits_{j=1}^{J}\left(x_j - \theta_j\right)^2 = \left[\underline{x} - \underline{\theta}\right]^T\left[\underline{x} - \underline{\theta}\right]. \tag{3.12}$$

Third, the gradient of a general vector $\underline{y}$ with respect to general vector $\underline{x}$ where $\underline{x} = [x_1, x_2, \ldots, x_L]^T$ and $\underline{y} = [y_1, y_2, \ldots, y_J]^T$ is expressed as follows [30]:

$$\nabla_{\underline{x}'}\underline{y} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_L} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_J}{\partial x_1} & \cdots & \frac{\partial y_J}{\partial x_L} \end{pmatrix}. \tag{3.13}$$

This derivation assumes three nodes, but it can be extended to N nodes. Given the measurement matrix $\underline{h}(\underline{x}_k) = \left[h_1(\underline{x}_k), h_2(\underline{x}_k), h_3(\underline{x}_k)\right]^T$ and the modified state vector $\underline{x}_k = \left[x_k, 0, 0, y_k, 0, 0\right]^T = \left[x_{1,k}, 0, 0, x_{4,k}, 0, 0\right]^T$, which contains position as the only state required for measurement linearization, the Jacobian can be written as

$$\mathrm{H} \triangleq \left.\frac{\partial \underline{h}\left[\underline{x}_k\right]}{\partial \underline{x}_k}\right|_{\underline{x}_k=\hat{\underline{x}}_{k|k-1}} \tag{3.14}$$

and expanded to

$$\begin{aligned} H &= \begin{pmatrix} \frac{\partial h_1\left[\underline{x}_k\right]}{\partial x_{1,k}} & 0 & 0 & \frac{\partial h_1\left[\underline{x}_k\right]}{\partial x_{4,k}} & 0 & 0 \\ \frac{\partial h_2\left[\underline{x}_k\right]}{\partial x_{1,k}} & 0 & 0 & \frac{\partial h_2\left[\underline{x}_k\right]}{\partial x_{4,k}} & 0 & 0 \\ \frac{\partial h_3\left[\underline{x}_k\right]}{\partial x_{1,k}} & 0 & 0 & \frac{\partial h_3\left[\underline{x}_k\right]}{\partial x_{4,k}} & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial\left(z_{0,1}-10\eta\log_{10}\left(d_{k,1}\left[\underline{x}_k\right]\right)\right)}{\partial x_{1,k}} & 0 & 0 & \frac{\partial\left(z_{0,1}-10\eta\log_{10}\left(d_{k,1}\left[\underline{x}_k\right]\right)\right)}{\partial x_{4,k}} & 0 & 0 \\ \frac{\partial\left(z_{0,2}-10\eta\log_{10}\left(d_{k,2}\left[\underline{x}_k\right]\right)\right)}{\partial x_{1,k}} & 0 & 0 & \frac{\partial\left(z_{0,2}-10\eta\log_{10}\left(d_{k,2}\left[\underline{x}_k\right]\right)\right)}{\partial x_{4,k}} & 0 & 0 \\ \frac{\partial\left(z_{0,3}-10\eta\log_{10}\left(d_{k,3}\left[\underline{x}_k\right]\right)\right)}{\partial x_{1,k}} & 0 & 0 & \frac{\partial\left(z_{0,3}-10\eta\log_{10}\left(d_{k,3}\left[\underline{x}_k\right]\right)\right)}{\partial x_{4,k}} & 0 & 0 \end{pmatrix}. \end{aligned} \tag{3.15}$$

The measurement model of Eq. (3.9) is a scalar model. We next convert this to a vector model by defining a vector of BS coordinates $\underline{\theta}_i = \left[a_i, 0, 0, b_i, 0, 0\right]^T$. Given Eq. (3.9), we redefine the distance vector to allow us to write the distances as quadratic forms using linear algebra resulting in

$$\underline{d}_k\left[\underline{x}_k\right] = \begin{pmatrix} d_{k,1} \\ d_{k,2} \\ d_{k,3} \end{pmatrix} = \begin{pmatrix} \sqrt{\left(\underline{x}_k-\underline{\theta}_1\right)^T G\left(\underline{x}_k-\underline{\theta}_1\right)} \\ \sqrt{\left(\underline{x}_k-\underline{\theta}_2\right)^T G\left(\underline{x}_k-\underline{\theta}_2\right)} \\ \sqrt{\left(\underline{x}_k-\underline{\theta}_3\right)^T G\left(\underline{x}_k-\underline{\theta}_3\right)} \end{pmatrix}, \tag{3.16}$$

where

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.17}$$

We can plug Eq. (3.16) and Eq. (3.17) into Eq. (3.15), apply the chain rule to take the partial differentials of the resultant matrix, and obtain the final Jacobian as

$$H = \begin{pmatrix} \frac{-10\eta\left(x_{k,1}-a_1\right)}{\ln(10)\left[\left(x_{k,1}-a_1\right)^2+\left(x_{k,4}-b_1\right)^2\right]} & 0 & 0 & \frac{-10\eta\left(x_{k,4}-b_1\right)}{\ln(10)\left[\left(x_{k,1}-a_1\right)^2+\left(x_{k,4}-b_1\right)^2\right]} & 0 & 0 \\ \frac{-10\eta\left(x_{k,1}-a_2\right)}{\ln(10)\left[\left(x_{k,1}-a_2\right)^2+\left(x_{k,4}-b_2\right)^2\right]} & 0 & 0 & \frac{-10\eta\left(x_{k,4}-b_2\right)}{\ln(10)\left[\left(x_{k,1}-a_2\right)^2+\left(x_{k,4}-b_2\right)^2\right]} & 0 & 0 \\ \frac{-10\eta\left(x_{k,1}-a_3\right)}{\ln(10)\left[\left(x_{k,1}-a_3\right)^2+\left(x_{k,4}-b_3\right)^2\right]} & 0 & 0 & \frac{-10\eta\left(x_{k,4}-b_3\right)}{\ln(10)\left[\left(x_{k,1}-a_3\right)^2+\left(x_{k,4}-b_3\right)^2\right]} & 0 & 0 \end{pmatrix}. \tag{3.18}$$

Recall that the model for the state of the mobile node is linear and therefore does not require a Jacobian for the EKF algorithm.

IV. THE EXTENDED KALMAN FILTER ALGORITHM

In this chapter, we summarize the general equations for the EKF we use in our solution of the mobility estimation problem. The following is the notation for this section. The discrete time index is denoted t. A "hat" above a symbol denotes an estimate (e.g., $\underline{\hat{x}}(t)$). A tilde above a symbol is used to denote an error or error covariance (e.g., $\underline{\tilde{x}}(t \mid t-1)$ or $\tilde{\tilde{P}}(t \mid t-1)$). The notation $\underline{\tilde{x}}(t \mid t-1)$ is read "the error in the states at time step t, given data up to time step $t-1$." The double tilde on $\tilde{\tilde{P}}(t \mid t-1)$ indicates an error covariance matrix.

The EKF is a state space nonlinear state estimator that provides estimates of the state vector at each discrete time step t. It is the optimal least squares estimator for our model. The EKF is an extension of the KF, a wholly linear estimator, because it handles the nonlinear Gauss-Markov model. The EKF development in [15] and [31] are closely followed in this thesis in order to develop the key equations needed for proper implementation. A block diagram of the UGV node and the EKF is illustrated in Figure 5 and Figure 6. All the equations in Figure 6 appear in the text.

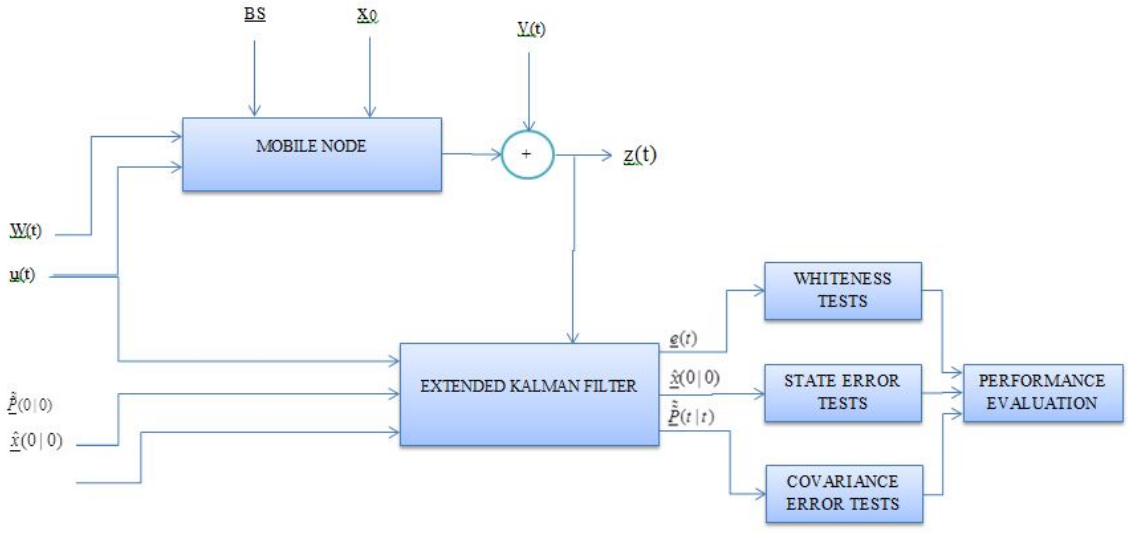

Figure 5. Signal flow block diagram of the mobile node model, EKF, and performance evaluation techniques along with input and outputs.

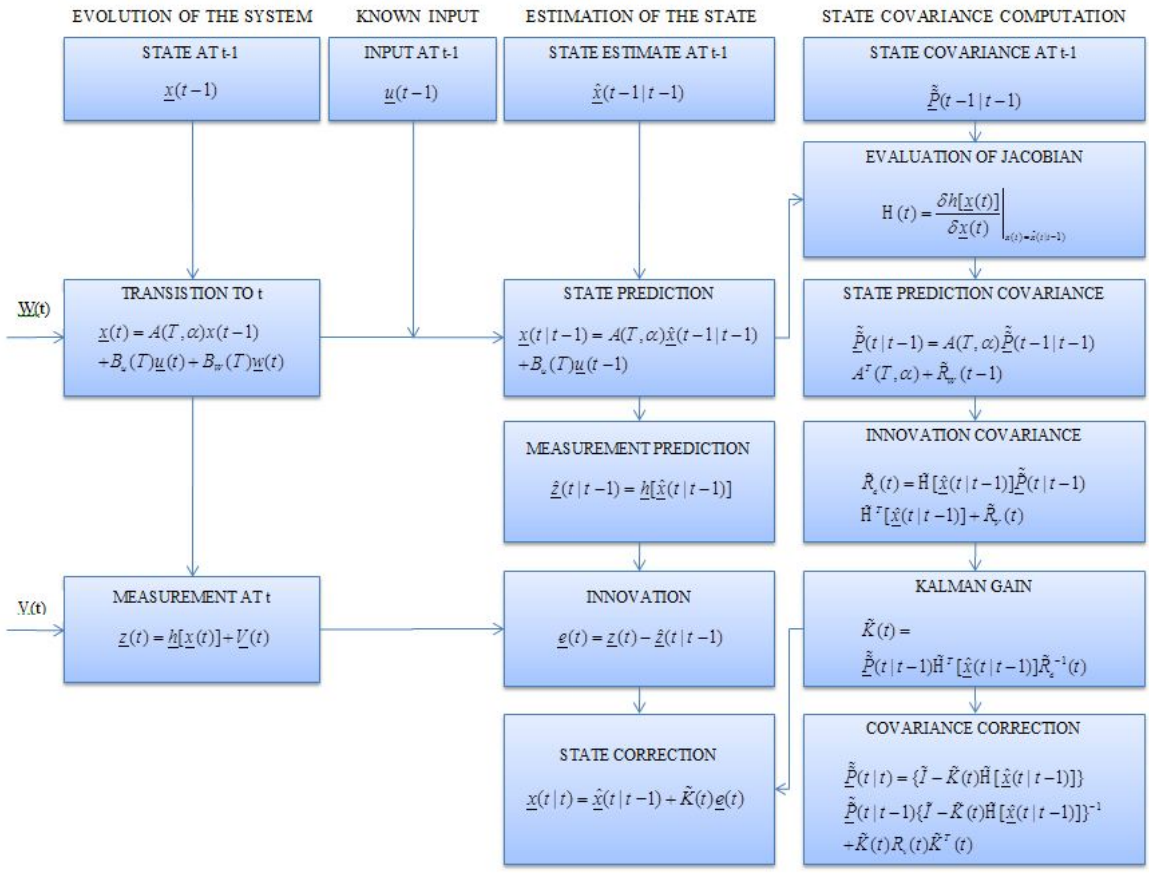

Figure 6. Flow diagram depicting the implementation of a discrete-time EKF algorithm for the UGV-DTN. The construction of the flow chart follows [32].

A. DISCRETE-TIME NONLINEAR GAUSS-MARKOV MODEL

Given state vector $\underline{x}(t)$; initial state vector $\underline{x}(0)$; system matrices $A(.)$, $B_u(.)$, and $B_w(.)$; system input vector $\underline{u}(t)$; and process noise $\underline{w}(t)$; we can write the state propagation model as follows (see Eq. (3.7)):

$$\underline{x}(t)=A(T,\alpha)\underline{x}(t-1)+B_u(T)\underline{u}(t)+B_w(T)\underline{w}(t). \tag{4.1}$$

Give the system output measurement vector $\underline{z}(t)=\left[z_1(t),z_2(t),z_3(t)\right]^T$, nonlinear function $h(.)$, and measurement noise vector $\underline{v}(t)$, we can write the measurement propagation as follows:

$$\underline{z}(t)=h\left[\underline{x}(t)\right]+\underline{v}(t). \tag{4.2}$$

Note that for our mobility problem $\underline{w}(t) = \left[w_x(t), w_y(t) \right]^T$ and $\underline{v}(t) = \left[v_1(t), v_2(t), v_3(t) \right]^T$ are zero-mean white Gaussian noise sequences with covariances $\tilde{R}_w$ and $\tilde{R}_v$, and distributions $w(t) \sim N\left[0, \tilde{R}_w\right]$ and $v(t) \sim N\left[0, \tilde{R}_v\right]$ [15].

B. DISCRETE-TIME EXTENDED KALMAN FILTER ALGORITHM

Given the nonlinear Gauss-Markov model for the mobile node, the discrete-time EKF algorithm is shown in flow-diagram in Figure 6. The derived EKF algorithm equations are summarized as follows [15], [31]:

1. Prediction

The state prediction step is

$$\underline{\hat{x}}(t \mid t-1) = A(T, \alpha)\underline{\hat{x}}(t-1 \mid t-1) + B_u(T)\underline{u}(t-1) \tag{4.3}$$

and the state error covariance step is

$$\tilde{\tilde{P}}(t \mid t-1) = A(T, \alpha)\tilde{\tilde{P}}(t-1 \mid t-1)A^T(T, \alpha) + \tilde{R}_W(t-1). \tag{4.4}$$

In the prediction step, we create two quantities: (1) First, we predict the next estimate of the state vector by propagating the state estimate from the previous time step through the system model (Eq [4.3]). (2) The predicted state estimation error is defined as $\underline{\tilde{x}}(t \mid t-1) \equiv \underline{x}(t) - \underline{\hat{x}}(t \mid t-1)$; and the predicted state error covariance is defined as $\tilde{\tilde{P}}(t \mid t-1) \equiv \mathrm{cov}\left[\underline{\tilde{x}}(t \mid t-1) \right]$. We predict the next estimate of the state error covariance by propagating the state estimation error covariance from the previous time step through the system matrix $A(T, \alpha)$ and adding the process noise covariance.

2. Innovation

The innovation step is

$$\underline{e}(t) = \underline{z}(t) - \underline{\hat{z}}(t \mid t-1) \tag{4.5}$$

and the innovation covariance step is

$$\tilde{R}_e(t) = \tilde{\mathrm{H}}[\underline{\hat{x}}(t \mid t-1)]\tilde{\tilde{\underline{P}}}(t \mid t-1)\tilde{\mathrm{H}}^T[\underline{\hat{x}}(t \mid t-1)] + \tilde{R}_V(t). \tag{4.6}$$

The innovations vector is the difference between the current measurement and the last estimate of the measurement vector given data up to time $t-1$. The innovations represent new information available to the EKF since the last state update; and they provide the key information we can use to ensure that the filter converges to a useful state estimate. Once the innovations vector is computed, we calculate the next value of the innovations covariance matrix using Eq. (4.6).

3. Gain

The Kalman gain is

$$\tilde{K}(t) = \tilde{\tilde{\underline{P}}}(t \mid t-1)\tilde{\mathrm{H}}^T[\underline{\hat{x}}(t \mid t-1)]\tilde{R}_e^{-1}(t). \tag{4.7}$$

The Kalman gain matrix provides the key factor we use in the next step to update the state estimate in a direction toward minimizing the mean square error. A small value of the Kalman gain indicates that from the filter "believes" (places a large weight on) the latest model predictions; whereas, a large gain indicates that the filter "believes" (places a large weight on) the latest measurements [15].

4. Correction

The state correction step is

$$\underline{x}(t \mid t) = \underline{\hat{x}}(t \mid t-1) + \tilde{K}(t)\underline{e}(t) \tag{4.8}$$

and the state error covariance estimator correction step is

$$\begin{aligned}\tilde{\tilde{\underline{P}}}(t \mid t) = &\{\tilde{I} - \tilde{K}(t)\tilde{\mathrm{H}}[\underline{\hat{x}}(t \mid t-1)]\}\tilde{\tilde{\underline{P}}}(t \mid t-1)\{\tilde{I} - \tilde{K}(t)\tilde{\mathrm{H}}[\underline{\hat{x}}(t \mid t-1)]\}^{-1} \\ &+ \tilde{K}(t)R_v(t)\tilde{K}^T(t).\end{aligned} \tag{4.9}$$

This is the key step in the EKF. The filter updates (corrects) the last state estimate by adding to it the product of the Kalman gain matrix and the innovations vector. By doing so, it moves the state estimate in a direction that reduces the mean square error in the expected value sense. After many time steps, a properly tuned EKF

will converge toward the minimum mean square error estimate of the states. In this correction step, the EKF also updates (corrects) the state estimation error covariance matrix as in Eq. (4.9).

5. Initial Conditions

The state initial condition matrix is

$$\underline{\hat{x}}(0|0) = \left[\hat{x}_1(0|0), \hat{x}_2(0|0), \hat{x}_3(0|0), \hat{x}_4(0|0), \hat{x}_5(0|0), \hat{x}_6(0|0)\right]^T \qquad (4.10)$$

and the covariance initial condition matrix is

$$\tilde{\tilde{P}}(0|0) = \begin{pmatrix} \tilde{\tilde{P}}_{x_1}(0|0) & 0 & 0 & 0 & 0 & 0 \\ 0 & \tilde{\tilde{P}}_{x_2}(0|0) & 0 & 0 & 0 & 0 \\ 0 & 0 & \tilde{\tilde{P}}_{x_3}(0|0) & 0 & 0 & 0 \\ 0 & 0 & 0 & \tilde{\tilde{P}}_{x_4}(0|0) & 0 & 0 \\ 0 & 0 & 0 & 0 & \tilde{\tilde{P}}_{x_5}(0|0) & 0 \\ 0 & 0 & 0 & 0 & 0 & \tilde{\tilde{P}}_{x_6}(0|0) \end{pmatrix}. \qquad (4.11)$$

Note that initial condition values are chosen and explored in Chapter V.

6. Jacobian Matrix

The Jacobian is

$$\mathrm{H} \triangleq \left.\frac{\partial \underline{h}[\underline{x}_k]}{\partial \underline{x}_k}\right|_{\underline{x}_k = \underline{\hat{x}}_{k|k-1}}. \qquad (4.12)$$

The Jacobian matrix in Eq. (4.12) is used to linearize the nonlinear RSSI measurement equation in Eq. (4.2).

C. PERFORMANCE MEASURES FOR THE EKF

The following section summarizes methods for evaluating the performance of the EKF and its application to the UGV-DTN scenario. These results coupled with the theoretical points developed in the previous chapters lead to the proper evaluation and

adjustment, or tuning, of the EKF. Performance measures serve as a means to ensure all the statistics are valid and may be used as valid estimates.

1. Zero-Mean Test on the Innovations

A tuned EKF provides the optimal, or minimum MSE estimate of the state vector. The innovation sequence is used for evaluating performance. A necessary and sufficient condition for the EKF to be optimal is that the innovations sequence must be zero mean and white [15]. If we assume that the innovations $\underline{e}(t) = \underline{z}(t) - \underline{\hat{z}}(t \mid t-1)$ are ergodic and Gaussian, we can use the sample mean as a test statistic in a zero-mean hypothesis test. The ith component of the mean of $\underline{e}(t) = \left[e_1(t), \ldots, e_p(t) \right]$ is given by:

$$\hat{m}_e(i) = \frac{1}{N} \sum_{t=1}^{N} e_i(t) \tag{4.13}$$

for $i = 1, 2, \ldots, p$, where $\hat{m}_e(i) \sim N\left(m_e, R_e(i)/N\right)$, p is the number of measurements or components in $\underline{e}(t)$, and N is the number of samples in the innovations sequence. The hypotheses in the hypothesis tests H_0 and H_1 are

$$H_0 : m_e = 0 \tag{4.14}$$

$$H_1 : m_e \neq 0. \tag{4.15}$$

At the significance level α_H, the probability of rejecting the null hypothesis H_0 is given by:

$$P\left(\left| \frac{\hat{m}_e(i) - m_e(i)}{\sqrt{R_e(i)/N}} \right| > \left| \frac{\tau_i(i) - m_e(i)}{\sqrt{R_e(i)/N}} \right| \right) = \alpha_H, \tag{4.16}$$

where $\hat{R}_e(i)$ is the sample variance (assuming ergodicity) is given by:

$$\hat{R}_e(i) = \frac{1}{N} \sum_{t=1}^{N} e^2(t). \tag{4.17}$$

Given significance level $\alpha_H = .05$ or 5%, the hypothesis test threshold is [15]

$$\tau_i = 1.96\sqrt{\frac{\hat{R}_e(i)}{N}}. \tag{4.18}$$

The zero-mean hypothesis test on each component of the innovation e_i is denoted by:

$$\hat{m}_e(i) \underset{<H_0}{\overset{>H_1}{}} \tau_i. \tag{4.19}$$

Practical implementation of the zero-mean whiteness test is achieved by plotting the innovation time series $e_i(t)$ along with the positive and negative threshold values on the same plot. The number of points that exceed the threshold are counted, divided by the total number of samples in the time series N, and compared to the significance level α_H to decide if the innovations can be deemed "white." The test has limited value unless the data are ergodic and Gaussian [31].

2. Innovations Whiteness Test

The innovations whiteness test is a measure of how well the EKF is tuned. Recall that a discrete-time stochastic process is "white" if the autocorrelation function is a Kronecker delta at lag zero [33]. This fact allows a practical statistical hypothesis test for whiteness. Assuming ergodicity, a test based on the normalized sample autocovariance function of the innovations sequence is

$$\hat{\rho}_e(i,k) = \frac{\hat{R}_e(i,k)}{\hat{R}_e(i)}, \tag{4.20}$$

where the ith component's innovation covariance is

$$\hat{R}_e(i,k) = \frac{1}{N}\sum_{t=k+1}^{N}\left[e_i(t) - \hat{m}_e(i)\right]\left[e_i(t+k) - \hat{m}_e(i)\right], \tag{4.21}$$

i is the index for the number of measurements $i = 1, 2, \ldots, p$, and k is the correlation lag index. For this test, the number of samples N represents the number of samples in the innovations sequence, over which the covariance is calculated, such that $k = 1, 2, \ldots, N$. Note that the sum from $t = k+1$ to N avoids the first sample, or the sample at zero lag,

which should equal one (the Kronecker delta) when we apply the hypothesis test described next.

It can be shown that the test statistic is Gaussian $\hat{\rho}_e(i,k) \sim N\left(0,1/N\right)$ for an asymptotically large $\hat{m}_e$ [31]; therefore, the 95% confidence interval estimate of $\hat{\rho}_e(i,k)$ is given by:

$$I_{\rho_e} = \hat{\rho}_e(i,k) \pm \frac{1.96}{\sqrt{N}}, \quad (N > 30). \tag{4.22}$$

Under the null hypothesis that the innovations $e_i(t)$ are white, the normalized autocovariance $\hat{\rho}_e(i,k)$ must lie within the interval I_{ρ_e} 95% of the time for H_0 to be accepted (i.e., to declare that the innovation is white).

In practice the test is implemented by plotting the normalized autocovariance $\hat{\rho}_e(i,k)$ over N lags, where $N > 30$, with the threshold $1.96/\sqrt{N}$ on the same plot. We then sum the number of samples that exceed the threshold, divide by N, and compare that fraction to the significance level to decide innovation whiteness [31].

3. Root Mean Squared State Estimation Error

The RMSE provides a measure of accuracy, or sufficiency, of the states of the estimator. The RSME evaluates the difference between the estimate and the true value within two standard deviations 2σ with 95% probability. With the definition of the state estimation error $\underline{\tilde{x}}_k$ defined as

$$\underline{\tilde{x}}_k \triangleq \underline{x}_k - \underline{\hat{x}}_{k|k-1}, \tag{4.23}$$

where $\underline{x}_k$ is the true state vector and $\underline{\hat{x}}_{k|k-1}$ is the estimated state vector, then the expected value of the inner product of the state estimation error $\underline{\tilde{x}}_k \triangleq \underline{x}_k - \underline{\hat{x}}_k$ is the estimator's variance or mean square error is

$$E\left[\underline{\tilde{x}}_k^{\,T}\underline{\tilde{x}}_k\right] = MSE\left(\underline{\tilde{x}}_k\right) \tag{4.24}$$

for the expectations in all cases [32].

The square root of the MSE, or RMSE,

$$\sigma_{\underline{\hat{x}}} \triangleq \sqrt{\mathrm{var}\left(\underline{\tilde{x}}_k\right)} \tag{4.25}$$

is the standard error, or standard deviation of the state estimation error. Practically the state RMSE is found by taking the difference between the true state vector $\underline{x}_k$ and the estimated state vector $\underline{\hat{x}}_{k|k-1}$ directly from the EKF and taking the square root, providing an accuracy up to two $\sigma_{\underline{\hat{x}}}$ with 95% probability [15], [32].

4. Weighted Sum Squared Residual

The innovations whiteness test above is valuable for evaluating the whiteness of one innovations component. Our system has multiple measurements; therefore, we need multiple innovations analysis. The weighted sum squared residual (WSSR) provides a method for whiteness testing over all of the innovations by aggregating innovations vector information $\underline{e}(l)$ into a single scalar test statistic. We define the WSSR as a scalar test statistic ρ as follows:

$$\rho(l) = \sum_{k=l-N+1}^{N} \underline{e}^T(k) R_e^{-1}(k) \underline{e}(k) \tag{4.26}$$

for $l \geq N$. Note that the WSSR is evaluated only for lag $l \geq N$, because we wish to inspect the error covariance at lags after which the transient in the covariance has settled down to a reasonable "steady state." Note also that the WSSR is calculated over a temporal window of N samples; and the window slides through the innovations data as the lag l increases. The hypothesis test for overall whiteness becomes

$$\rho(l) \underset{<H_0}{\overset{>H_1}{}} \tau, \tag{4.27}$$

where τ denotes the decision threshold. Under the null hypothesis, $\rho(l) \sim \chi^2(Np)$. However, for $Np > 30$, $\rho(l) \sim N(Np, 2Np)$ [31]. The probability of rejecting the null hypothesis at significance level α is

$$P\left(\left|\frac{\rho(l)-Np}{\sqrt{2Np}}\right|>\left|\frac{\tau-Np}{\sqrt{2Np}}\right|\right)=\alpha. \tag{4.28}$$

For a significance level of $\alpha=.05$, the threshold is

$$\tau = Np+1.96\sqrt{2Np}, \tag{4.29}$$

where p is the number of measurements and N is the number of covariance lag samples after which we evaluate the WSSR. Note that the value of N can be adjusted in the WSSR test [31].

Practical implementation of WSSR is achieved by plotting the WSSR for lags beyond N and plotting τ on the same plot. The number of WSSR samples that exceed the threshold are summed, divided by the total number of WSSR samples, and compared to the significance level α to determine the whiteness of the aggregated innovation information in order to evaluate overall whiteness [31].

5. Posterior Cramer-Rao Lower Bound

The PCRLB gives a lower bound on the achievable variance in the estimation of a parameter allowing the evaluation of quality. According to the PCRLB, the quantity related to the likelihood of the function must be smaller than the MSE corresponding to the estimator of the parameters. Therefore, the PCRLB gives a reference point from which to evaluate the estimator uncertainty. Assuming nonbiased estimators and nonrandom vector parameters, the PCRLB states that the covariance matrix of the state estimate error is bounded as follows:

$$P_{k|k}=E_{k,\underline{x}}\left[\left(\hat{\underline{x}}_{k|k}-\underline{x}_k\right)\left(\hat{\underline{x}}_{k|k}-\underline{x}_k\right)^T\right]\geq J_k^{-1}, \tag{4.30}$$

where J_k^{-1} the lower bound on the mean square of the estimate $\hat{\underline{x}}_{k|k}$. The Fisher Information Matrix (FIM) J_k is

$$J_k=E_{\underline{x}_k}\left\{\left[\nabla_{\underline{x}_k}\lambda\left(\underline{x}_k\right)\right]\left[\nabla_{\underline{x}_k}\lambda\left(\underline{x}_k\right)\right]'\right\}\Big|_{\underline{x}_k=\underline{x}_0}, \tag{4.31}$$

with gradient $\nabla_{\underline{x}}$ defined as in Eq. (3.13), the true value of the vector parameter $\underline{x}_k$ as $\underline{x}^*$, and the likelihood function as $\lambda(\underline{x}_k) = \ln p(\underline{x}^* \mid \underline{x}_k)$. The FIM is the contribution of the existing information to the data. Efficiency is achieved when the amount of extracted information is equal to the amount of the existing information. If the FIM is invertible (i.e., it is not singular), then the parameter is observable and sufficient information exists to allow estimation without ambiguity [15], [31]. Practical implementation of the PCRLB is achieved by taking the state estimation error covariance matrix of the EKF algorithm with the Jacobian evaluated at the true state $\underline{x}_k$ and plotting the resulting estimation error sequence along with estimated state $\hat{\underline{x}}_{k|k}$ RMSE on the same plot. The location PCRLB for our model is determined to be [16]

$$PCRLB = \sqrt{P_{k|k}(1,1) + P_{k|k}(4,4)}, \tag{4.32}$$

where $P_{k|k}(1,1)$ and $P_{k|k}(4,4)$ correspond to positions within the $P_{k|k}$ matrix.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION EXPERIMENT AND PERFORMANCE EVALUATION

In this section, we present the results of a simulation study, conducted in MATLAB [34], to demonstrate and validate the algorithms described earlier. We simulate a single mobile node traveling along a trajectory that includes abrupt maneuvers. We use a Gauss-Markov state space model for the node dynamics. Process noise is assumed to be zero. The measurements are constant power RSSI signals transmitted from fixed position base stations. We use the EKF derived in Chapter IV for state estimation, including node position coordinates in a two-dimensional spatial grid environment. Estimation performance is measured using zero mean whiteness tests on the innovations sequences, RMSE of the state estimates, WSSRs on the innovations, and the PCRLB.

A. CHOICES FOR THE SIMULATION AND EKF INITIAL PARAMETERS

1. Model simulation parameters

The parameters for Simulink are shown in Table 1.

Table 1. Simulation parameters for MATLAB implementation. The parameters follow from [16].

Discretization time step T	$0.5[s]$
Correlation coefficient α	0.6
Path loss index η	3
Base station transmission power $z_{0,i}$	90
Covariance σ_w^2 **of the noise** w_k	$0.5^2\left[m/s^2\right]^2$
Covariance σ_v^2 **of the noise** $v_{i,k}$	$4^2\left[dB\right]^2$
Maximum speed $V_{\max}$	$45[m/s]$
Transition probabilities $p_{i,i}$	0.8
Initial mode probabilities $\mu_{i,0}$	$1/M, i=1,\ldots,M, M=5$

The parameters are chosen such that the node behavior is realistic. Abrupt maneuvers are included to test the estimator's ability to adapt to rapid trajectory changes.

2. EKF Initial Conditions

The initial state and covariance estimates are given in Table 2.

Table 2. EKF initial conditions for MATLAB implementation.

Initial state estimate $\underline{\hat{x}}(0\mid 0)$	$\begin{pmatrix} 3400 \\ 5 \\ 0 \\ 8700 \\ 8 \\ 0 \end{pmatrix}$
Initial covariance estimate $\tilde{\tilde{P}}(0\mid 0)$	$\begin{pmatrix} 400^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 15^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 400^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 15^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5^2 \end{pmatrix}$

The rule of thumb for choosing the initial state estimate is as follows: we assume that we have reasonable a priori knowledge of the initial states of the UGV node because we deploy the nodes ourselves. The rule of thumb for choosing the initial covariance estimate is as follows: we use our engineering judgment to estimate the standard deviations of the node states based on our knowledge of the operational environment and node capabilities. Small values within the initial covariance matrix imply high levels of confidence in the initial state estimate. Conversely, large values place more emphasis on the ability of the state estimator to eventually converge to the proper solution.

B. SIMULATE THE COMMAND INPUT

The command input in the testing scenario is generated manually and is assumed to have zero process noise; thus, the input is deterministic. Short-time maneuvers are

followed by uniform motion similar to the methodology deployed in [16]. The discrete-time command processes $u_{x,k}$ and $u_{y,k}$ can change within the range $[-5,5]\left[m/s^2\right]$. The command process $\underline{u}_k$ in the filter is assumed to be a Markov chain, taking the values between the following discrete acceleration levels $M = M_x \times M_y = \{(0.0,0.0),(3.5,0.0),$ $(0.0,3.5),(0.0,-3.5),(-3.5,0.0)\}$ in the units of $\left[m/s^2\right]$. A plot of the command input processes $u_{x,k}$ and $u_{y,k}$ from the first order semi-Markov chain is illustrated in Figure 7. From the command input we expect the UGV node to turn twice, 150 and 200 seconds into the simulation. Knowledge of the turn times shown in Figure 7 allows easy interpretation of many of the figures to follow.

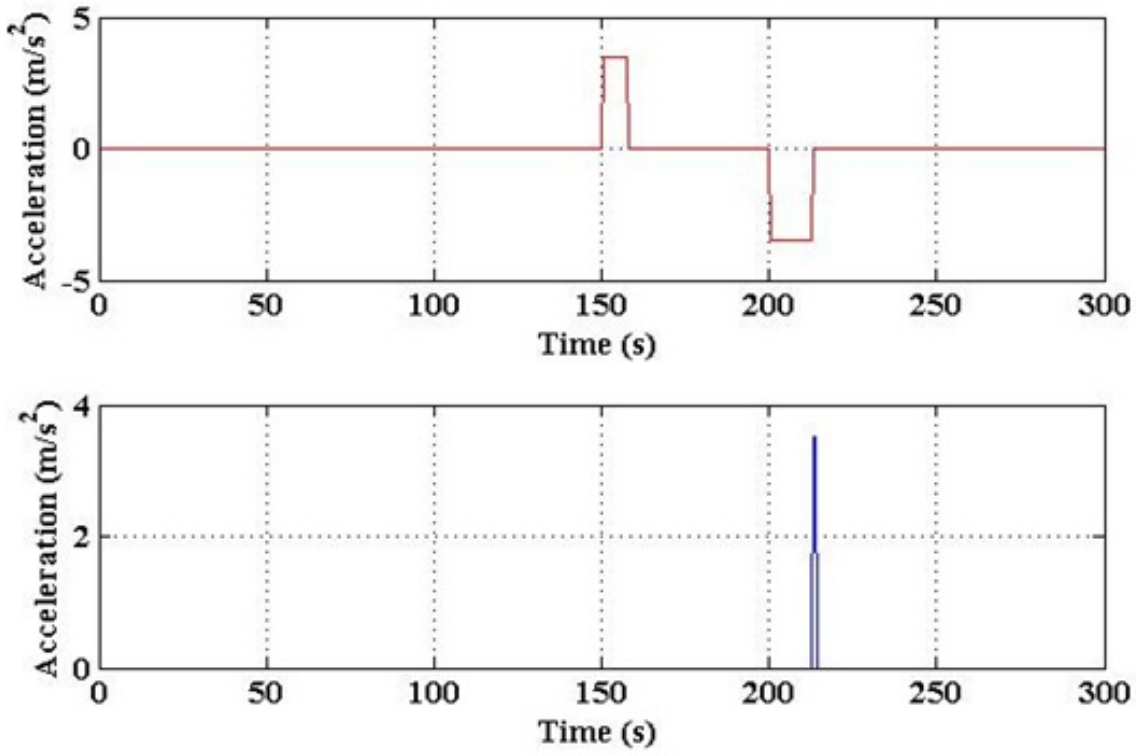

Figure 7. Command input processes $u_{x,k}$ and $u_{y,k}$ of the first order semi-Markov chain chosen for this experiment.

C. SIMULATE THE UNCERTAINTIES

White noise sequences are used to mimic the changing UGV-DTN node and the noisy signal measurement. The changing UGV-DTN node is modeled with zero mean, white Gaussian process noise $\underline{w}_k = \left[w_{x,k}, w_{y,k} \right]^T$, where $\underline{w}_{k,i} \sim N\left[0, \sigma_w^2 \right]$ for $i = 1,2$. The noisy signal measurement is modeled with zero mean, white Gaussian measurement noise $\underline{v}_k$, where $\underline{v}_{k,i} \sim N\left[0, \sigma_v^2 \right]$ for $i = 1,2,3$.

The trajectory for the UGV-DTN node was created deterministically by using zero process noise, essentially removing node acceleration uncertainty and simulating the ideal case. Even though we did not use process noise, an example of process noise over time is plotted with the two sigma bounds $\pm 2\sigma_w = \pm 1$ overlayed and is illustrated in Figure 8. The histogram of the process noise is presented in Figure 9. We see from the figure that the process noise has zero mean and variance equal to one. The distribution of values in the histogram appears Gaussian around zero.

The randomness of the RSSI comes from the randomness in the shadowing component modeled as measurement noise $\underline{v}_k$. The measurement noise over time is plotted with the two sigma bounds $\pm 2\sigma_v = \pm 8$ in Figure 10. The histogram of the measurement noise is presented in Figure 11. We see from the figure that the process noise has zero mean and variance equal to eight. The distribution of values in the histogram appears Gaussian around zero.

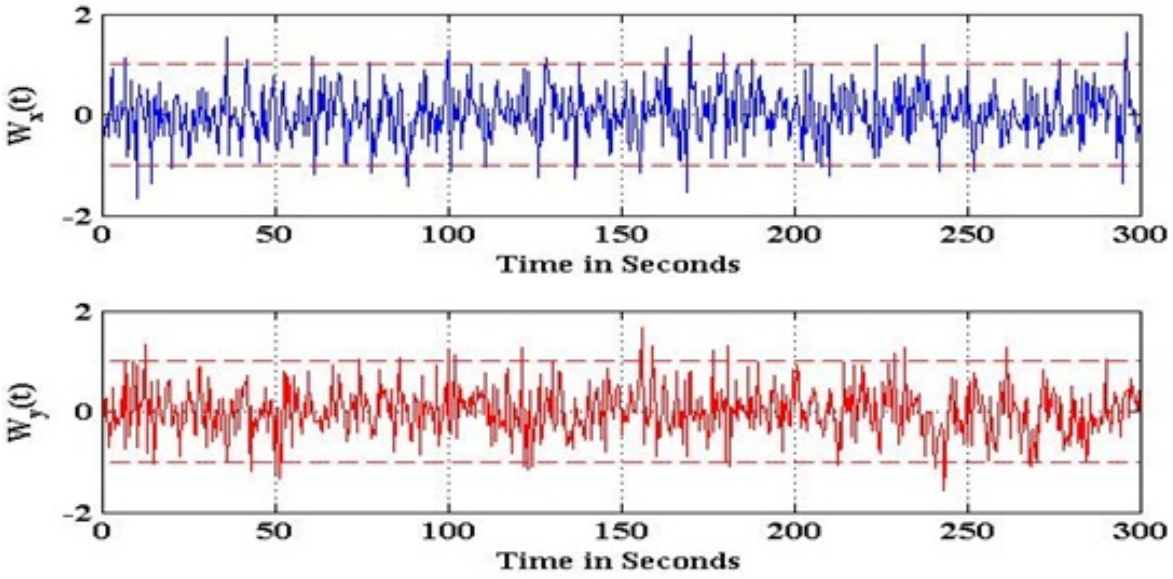

Figure 8. Process noise $\underline{w}_k = \left[w_{x,k}, w_{y,k} \right]^T$ of the UGV node over time with corresponding two sigma bounds $\pm 2\sigma_w = \pm 1$.

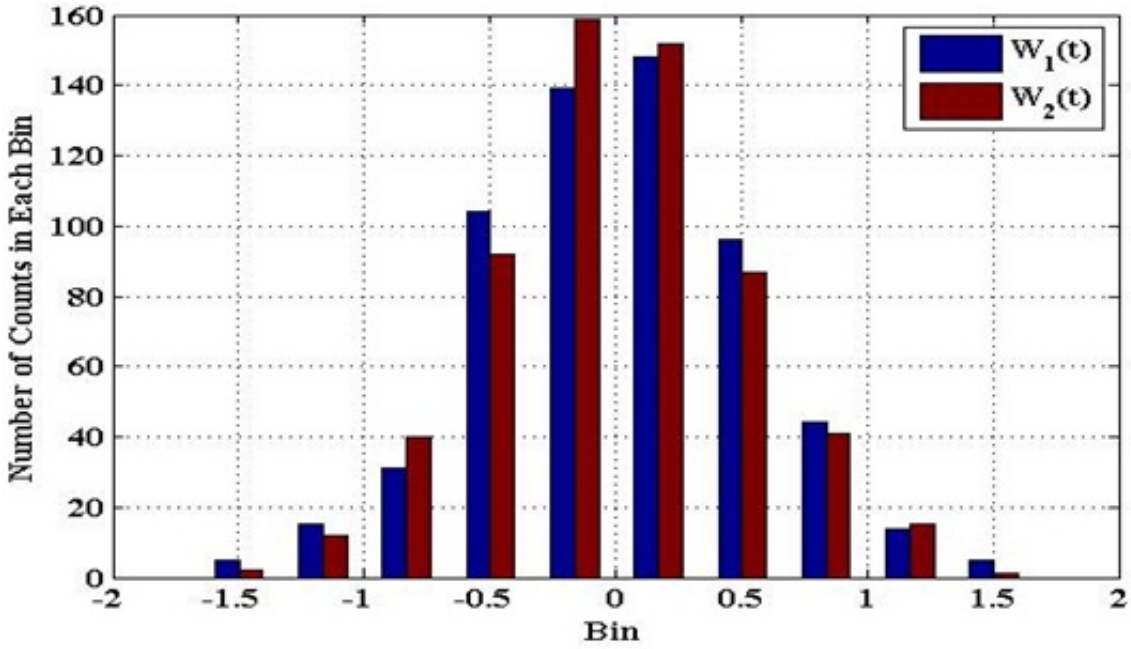

Figure 9. Histogram of the zero mean, white Gaussian process noise $\underline{w}_{k,i} \sim N\left[0, \sigma_w^2\right]$ for $i = 1, 2$.

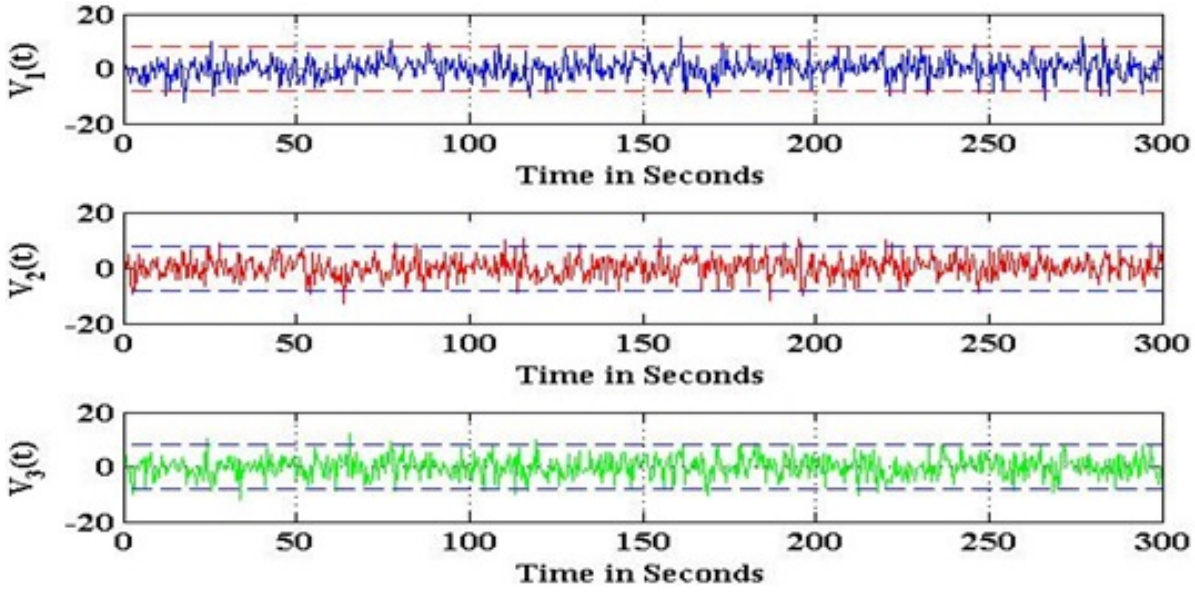

Figure 10. Measurement noise $\underline{v}_k = \left[v_1, v_2, v_3\right]^T$ of the UGV node over time with the corresponding two sigma bounds $\pm 2\sigma_v = \pm 8$.

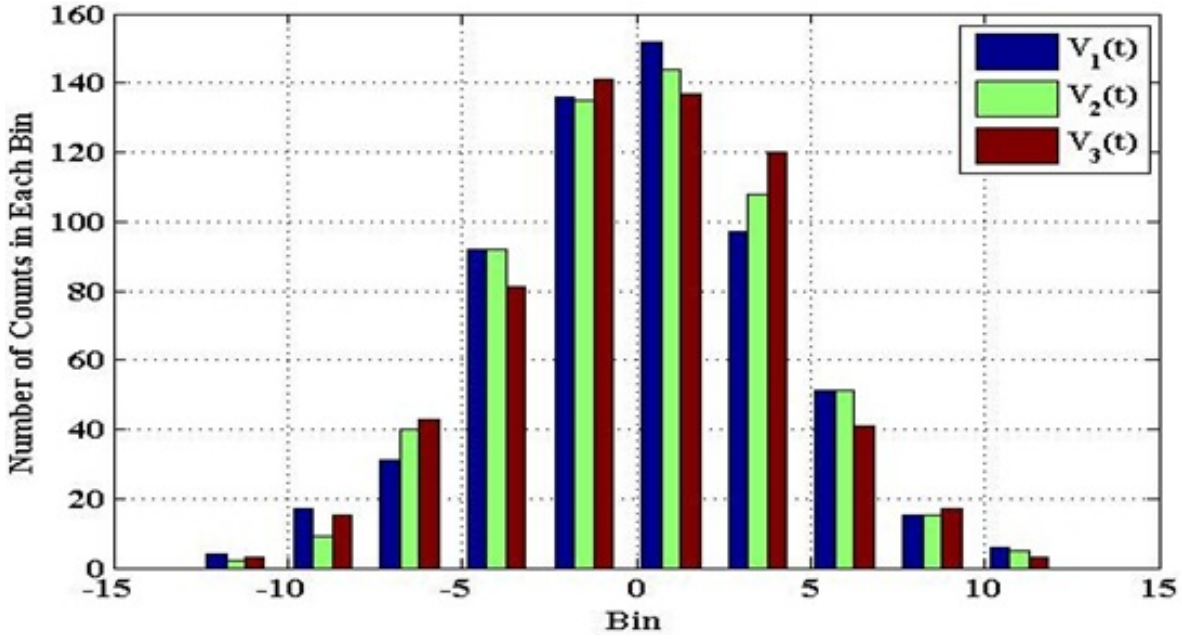

Figure 11. Histogram of the zero mean, white Gaussian measurement noise $\underline{v}_{k,i} \sim N\left[0, \sigma_v^2\right]$ for $i = 1, 2, 3$.

D. ESTIMATION OF STATES WITH THE EKF

A plot of the estimated track $\underline{\hat{x}}_k$ from the EKF overlayed on a plot of the actual trajectory to include base stations used for triangulation is shown in Figure 12. After the initial track errors during the transient state, the estimation settles into a trajectory that tracks closely to the actual trajectory. A plot of the estimated root mean speed $\hat{\dot{x}}_k = \sqrt{\hat{x}_{2,k}^2 + \hat{x}_{5,k}^2}$ and x and y velocity, $\hat{x}_{2,k}$, overlayed on a plot of the actual root mean speed $\dot{x}_k = \sqrt{x_{2,k}^2 + x_{5,k}^2}$ and x and y velocity, $x_{2,k}$ and $x_{5,k}$, is illustrated in Figure 13. The initial velocity errors settle after about 40 seconds of transient behavior and closely track the true velocity.

The simulated measurements $\underline{z}_k = \left[z_{1,k}, z_{2,k}, z_{3,k} \right]^T$ from the Gauss-Markov model are plotted and overlayed on the estimated measurements $\underline{\hat{z}}_k = \left[\hat{z}_{1,k}, \hat{z}_{2,k}, \hat{z}_{3,k} \right]^T$ from the EKF in Figure 14. The estimated measurements $\underline{\hat{z}}_k$ carry a small, fluctuating variance from the simulated measurements $\underline{z}_k$. The error between the estimated and actual states $\underline{\tilde{x}}_k = \underline{\hat{x}}_k - \underline{x}_k$ over time is presented in Figure 15. The errors $\underline{\tilde{x}}_k$ are shown to be acceptable in that they are approximately zero mean and Gaussian in distribution. The errors are shown to be zero-mean and lie within the two-sigma bounds an appropriate amount of the time at only three tenths of a percent deviation each.

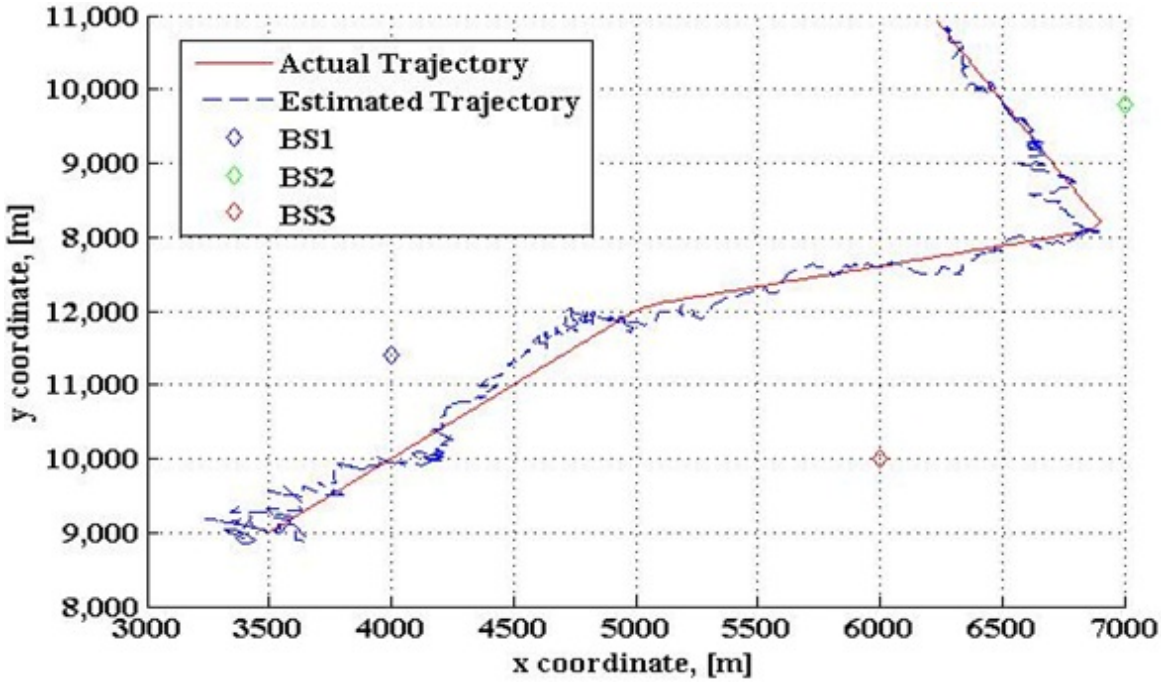

Figure 12. Estimated track, simulated track, and locations of base stations transmitting RSSI signals used for triangulation of the UGV node.

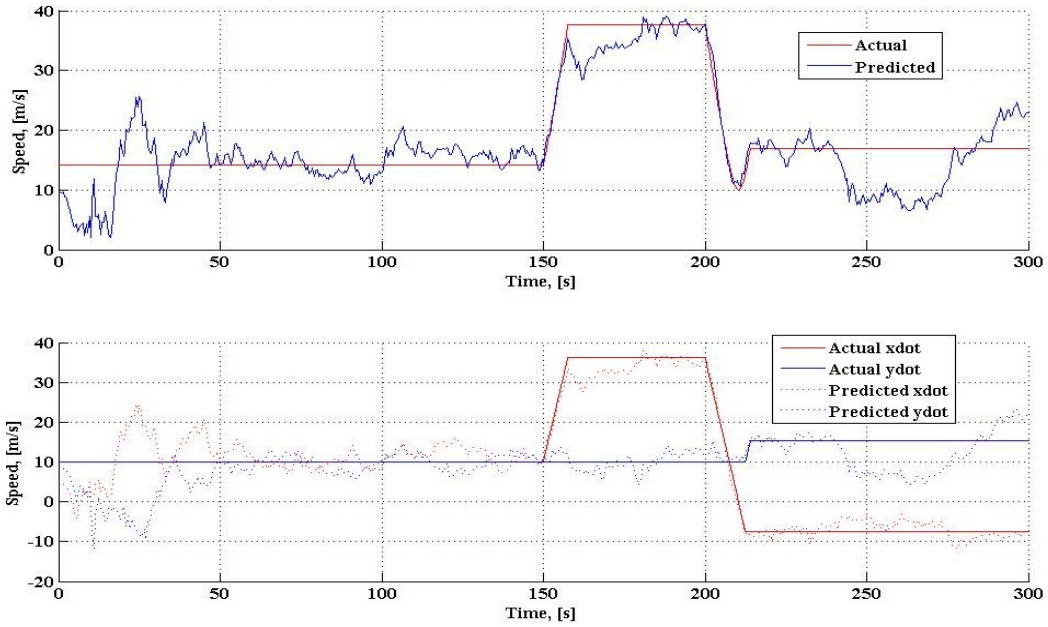

Figure 13. Speed plots of the UGV node. Top plot: estimated root mean speed $\hat{\dot{x}}_k = \sqrt{\hat{x}_{2,k}^2 + \hat{x}_{5,k}^2}$ and actual root mean speed $\dot{x}_k = \sqrt{x_{2,k}^2 + x_{5,k}^2}$ of the node. Bottom plot: estimated x and y velocity, $\hat{x}_{2,k}$ and $\hat{x}_{5,k}$, and actual x and y velocity, $x_{2,k}$ and $x_{5,k}$, of the node.

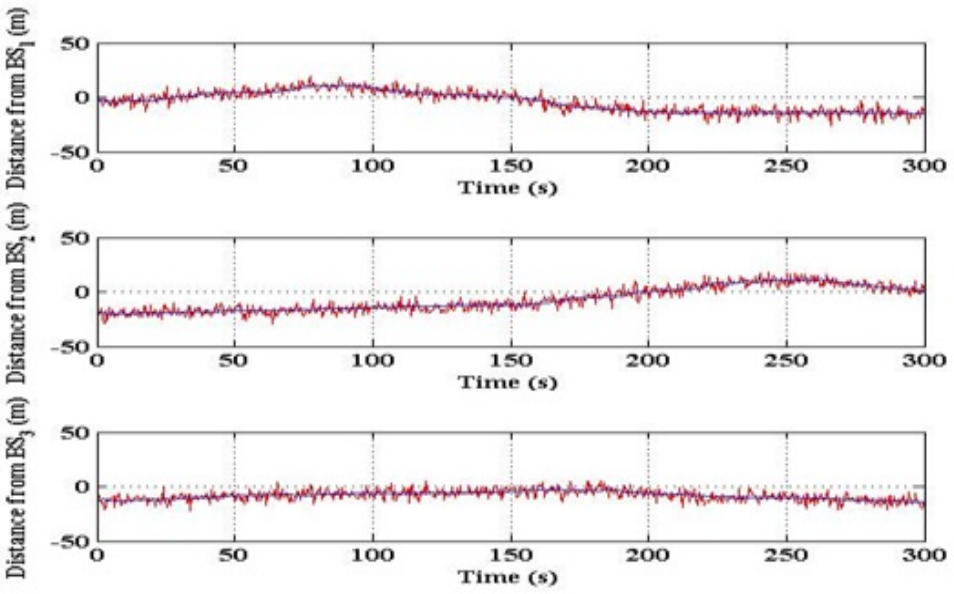

Figure 14. Noisy RSSI measurements $\underline{\hat{z}}_k = \left[\hat{z}_{1,k}, \hat{z}_{2,k}, \hat{z}_{3,k}\right]^T$ of the UGV node plotted against the true measurements $\underline{z}_k = \left[z_{1,k}, z_{2,k}, z_{3,k}\right]^T$ of the UGV node.

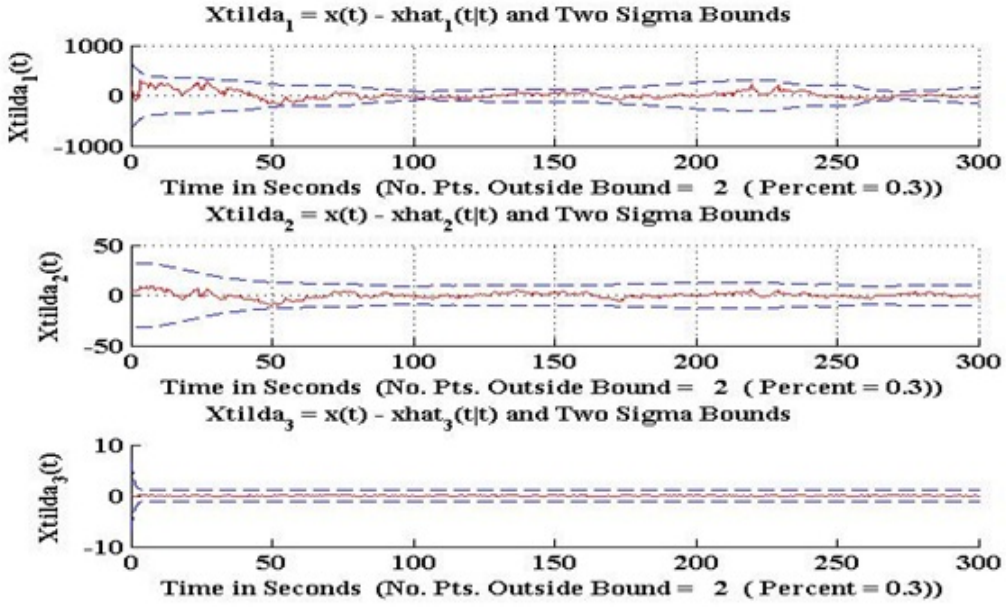

Figure 15. Error between the estimated states and the actual states $\underline{\tilde{x}}_k = \underline{\hat{x}}_k - \underline{x}_k$ of the UGV nodes and their respective two sigma bounds plotted over time. Top row corresponds to the position, middle row corresponds to the velocity, and bottom row corresponds to acceleration of the UGV node.

E. PERFORMANCE AND TUNING OF THE EKF

Now we examine the performance of the EKF using the methods described earlier. First we plot the innovations $\underline{e}_k = \underline{z}_k - \underline{\hat{z}}_{k|k-1}$ from the EKF along with their "two-sigma bounds" in Figure 16. This plot shows that the innovations exhibit good behavior through the data record with non-zero means and magnitudes appropriately at only 0.3% deviation each beyond the bounds (noted at the bottom of each plot).

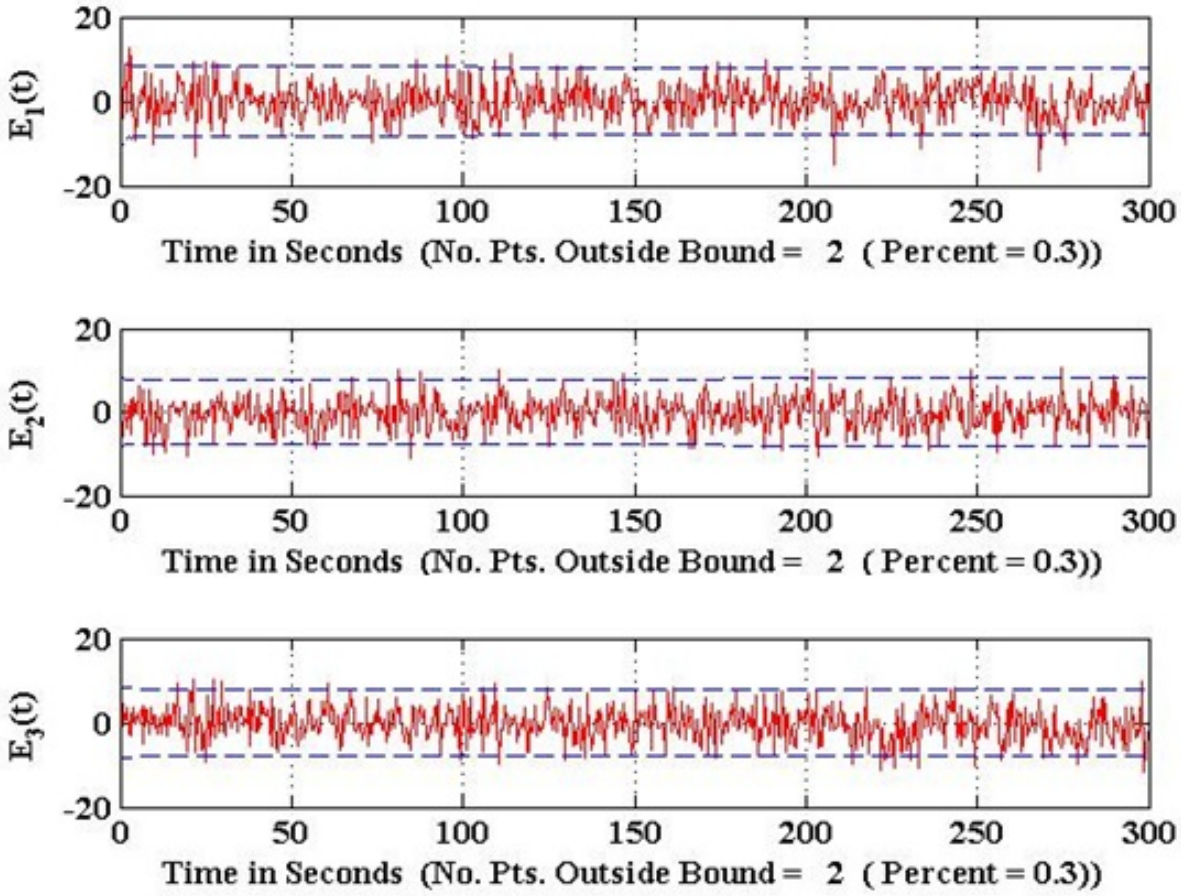

Figure 16. Innovations sequences $\underline{e}_k = \underline{z}_k - \underline{\hat{z}}_{k|k-1}$ of the UGV node and corresponding two-sigma bounds plotted over time.

The whiteness test plots for the three components of the innovations vector $\underline{e}_k$ are shown in Figure 17, Figure 18, and Figure 19. It can be seen that all innovations are white. This indicates that the autocovariances converge to a value within the two-sigma

bounds. In addition, there are not too many samples outside the bounds at small lags, meeting the significance criterion of 5%. The deviation beyond the two-sigma bounds is noted at the bottom of each plot.

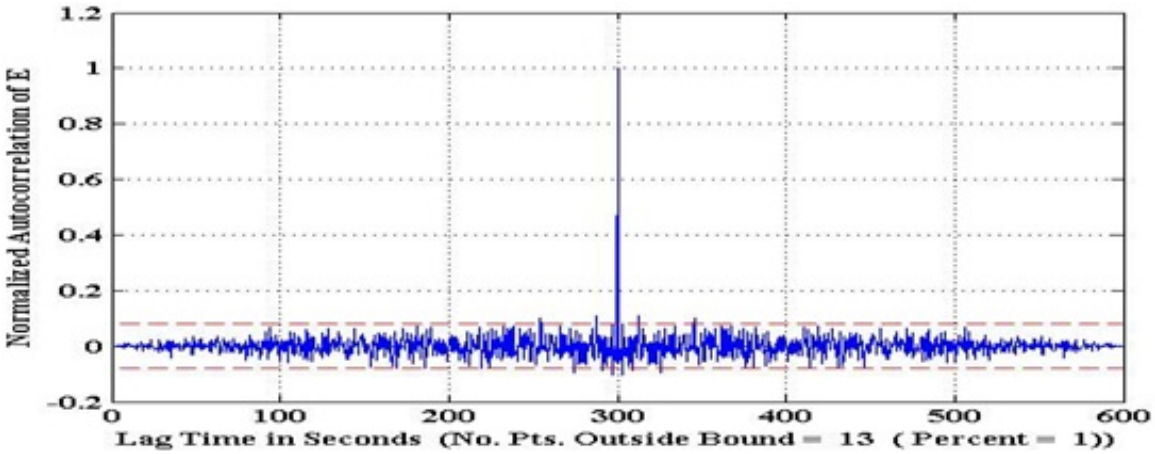

Figure 17. Whiteness test for the innovations $\underline{e}_1(k) = \underline{z}_1(k) - \hat{\underline{z}}_1(k \mid k-1)$ on the measurement from the first base station. Positive and negative lags with zero lag appearing in the middle of the plot at sample 300.

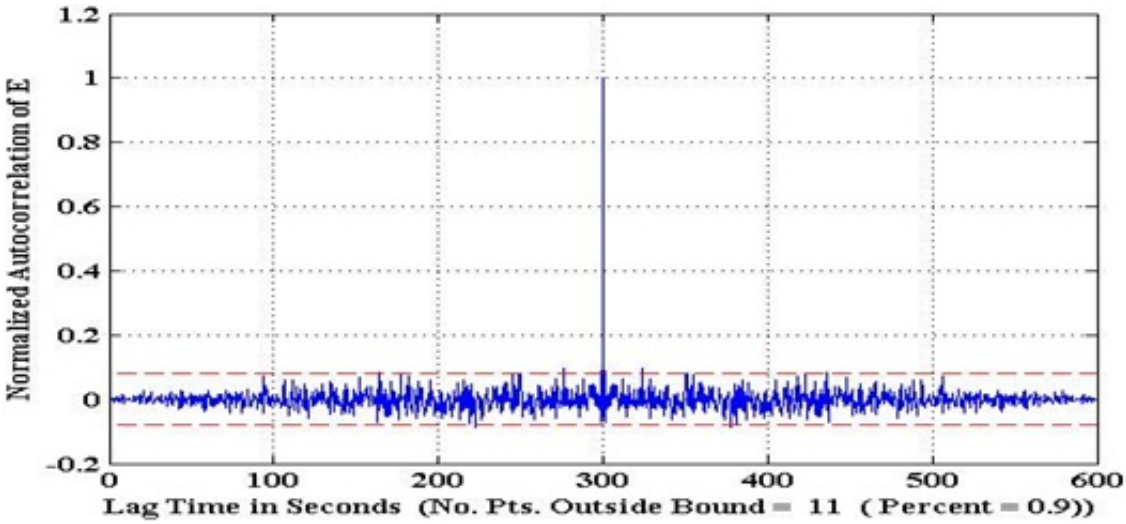

Figure 18. Whiteness test for the innovations $\underline{e}_2(k) = \underline{z}_2(k) - \hat{\underline{z}}_2(k \mid k-1)$ on the measurement from the second base station. Positive and negative lags with zero lag appearing in the middle of the plot at sample 300.

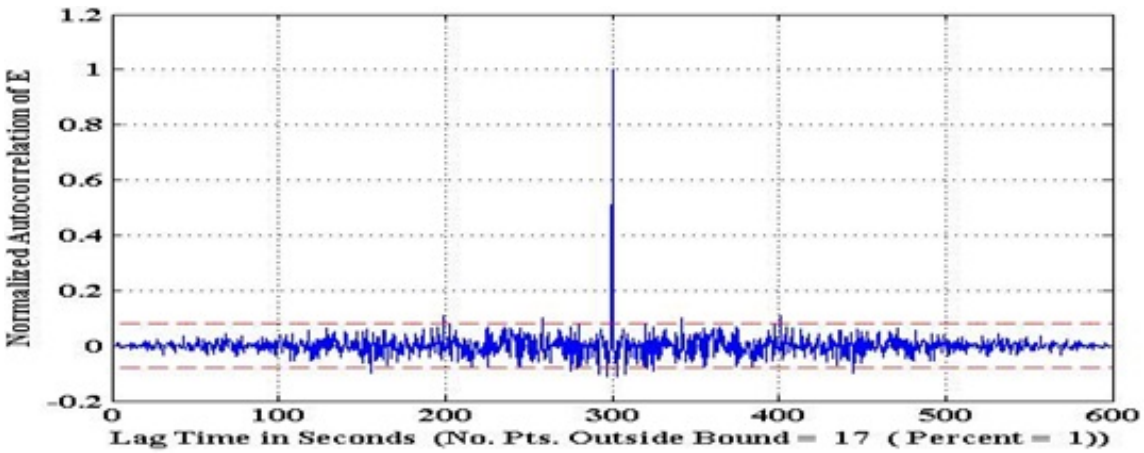

Figure 19. Whiteness test for the innovations $\underline{e}_3(k) = \underline{z}_3(k) - \hat{\underline{z}}_3(k \mid k-1)$ on the measurement from the third base station. Positive and negative lags with zero lag appearing in the middle of the plot at sample 300.

In order to assess the closeness of the estimated trajectory $\hat{\underline{x}}_k$ to a given trajectory $\underline{x}_k$, we ensemble averaged the RMSE and PCRLB realizations to smooth RMSE and PCRLB estimates. The ensemble average is taken over $N_{MC} = 100$ Monte Carlo runs of the UGV-DTN mobile node scenario in MATLAB. The position and speed RMSE overlayed with the plot of the PCRLB for position and speed are shown in Figure 20 and Figure 21, respectively. In addition, the error between the state RMSE position and velocity and the PCRLB position and velocity $\tilde{\underline{x}}_k = \hat{\underline{x}} - \underline{x}$ is examined over 100 Monte Carlo runs in Figure 22. The errors in position are shown to be less than 100 meters on average. The average errors in velocity are shown to be less than 5 meters per second on average. This indicates that at any time the EKF estimated the position to within 100 meters and velocity to within 5 meters per second of the best possible estimate represented by the PCRLB on average. Given that the initial covariance value was set with an expected uncertainty of 400 meters in position and 15 meters per second in velocity, the EKF performed quite sufficiently.

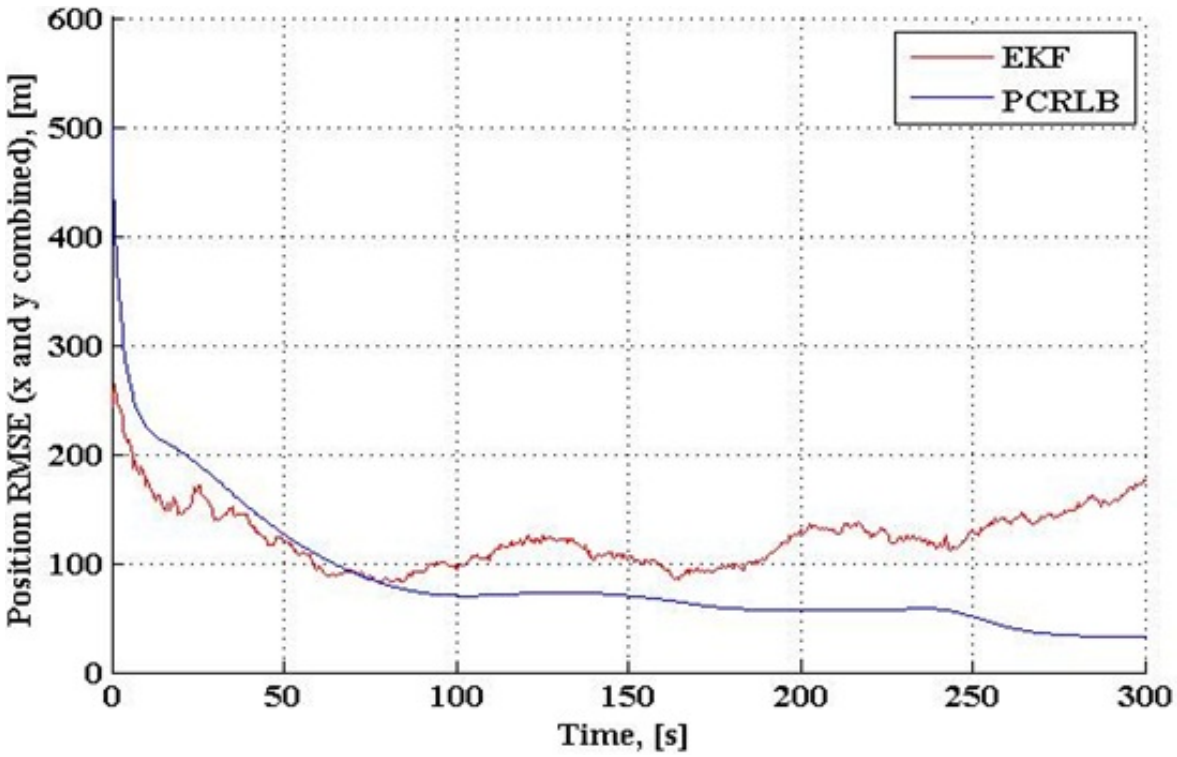

Figure 20. Ensemble average of the position RMSE plotted with the ensemble average of the position PCRLB of the UGV node over 100 runs.

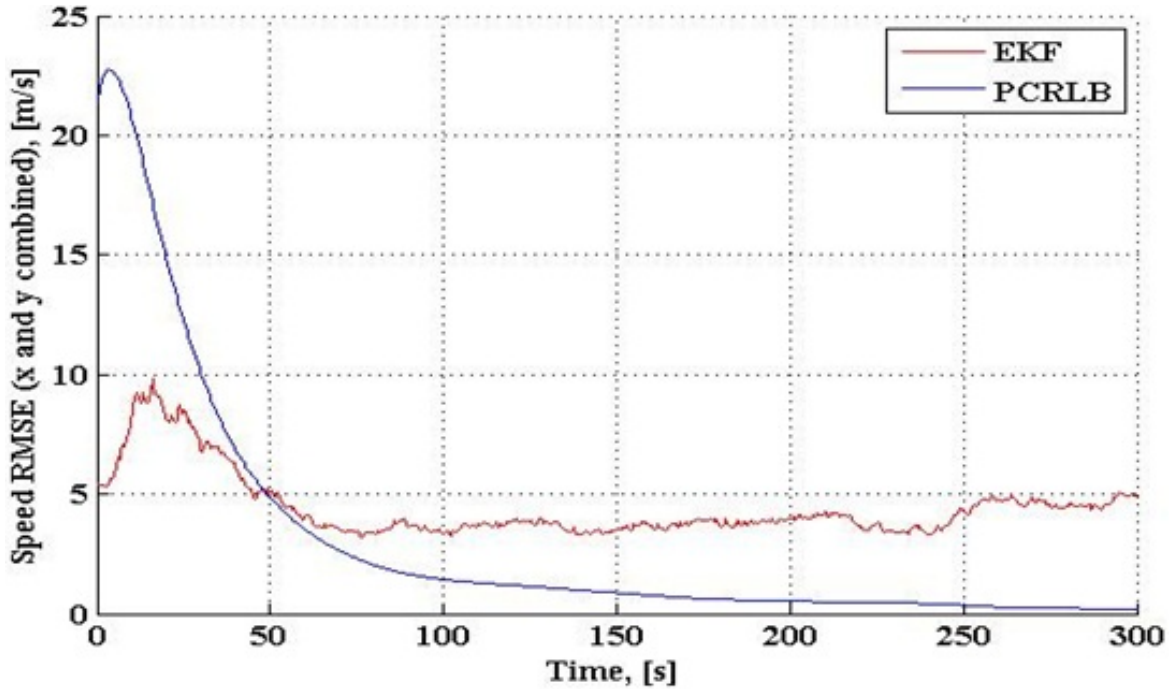

Figure 21. Ensemble average of the velocity RMSE plotted with the ensemble average of the velocity PCRLB of the UGV node over 100 runs.

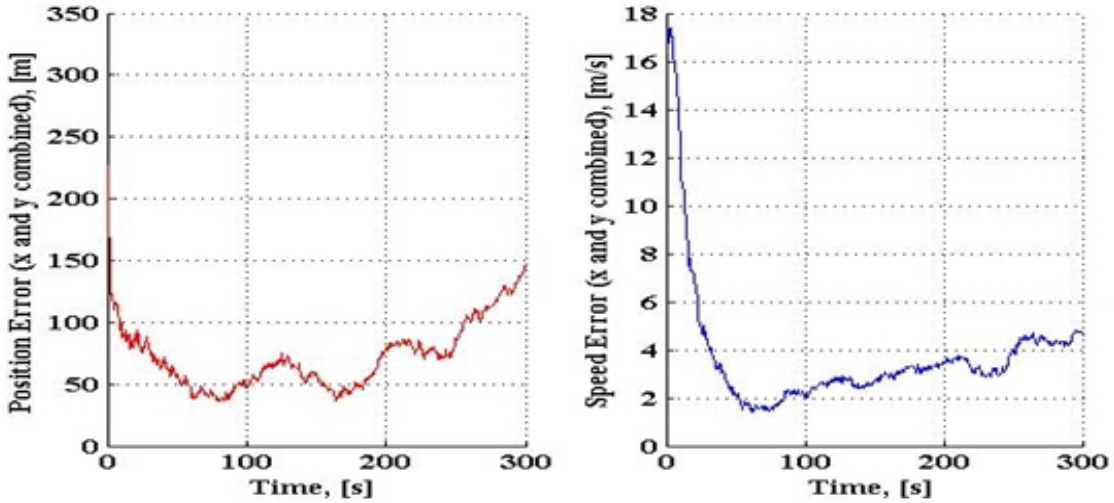

Figure 22. Error between the state RMSE and the PCRLB $\underline{\tilde{x}}_k = \underline{\hat{x}} - \underline{x}$ over 100 Monte Carlo runs. Left plot: difference between the ensemble average of position the RMSE and the ensemble average of the PCRLB of the UGV node over 100 runs illustrated in Figure 20. Right plot: difference between the ensemble average of the velocity RMSE and the ensemble average of the PCRLB of the UGV node over 100 runs illustrated in Figure 21.

We examine the WSSR to further explore the EKF performance for the innovations vector $\underline{e}_k$ in Figure 23. From Figure 23, it can be seen that the WSSR never exceeds its threshold, so by this criterion, we declare the EKF to be tuned and the overall performance to be acceptable.

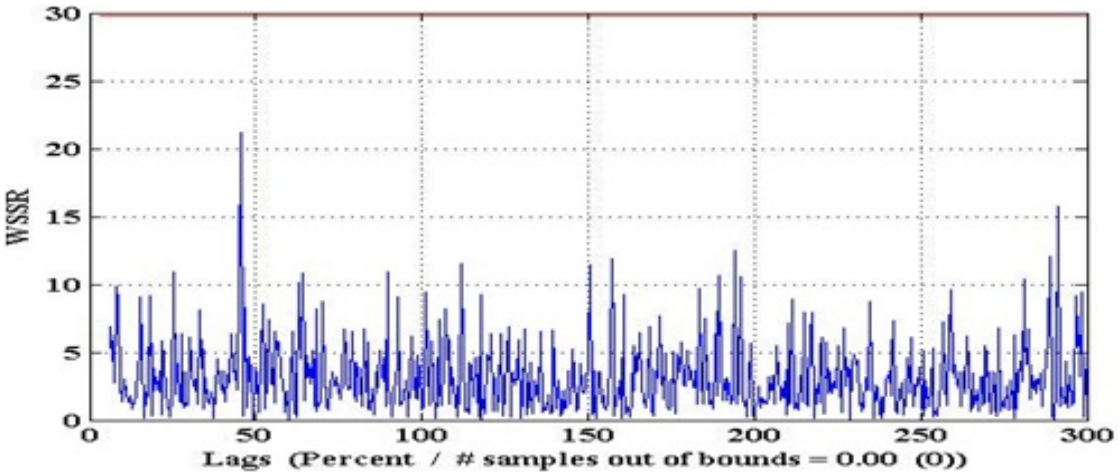

Figure 23. Aggregated innovations vector information WSSR threshold in red plotted against the aggregated innovations vector information WSSR sequence in blue.

It is important to note that EKF performance is highly sensitive to the choice of initial conditions on the state vector and state covariance. The more prior knowledge one has of the operational environment and node behavior, the better choices one can make for the initial conditions. The closer the initial state vector and state covariance matrix are to the true state vector and state covariance matrix, the more rapidly the EKF will converge to the proper solution.

The EKF initial conditions determine the reaction of the UGV-DTN node in order to converge to the desired states. That is to say that the more confidence you have in your initial state conditions, implying low uncertainty or initial covariance levels, the slower the UGV-DTN initially reacts to changes in the desired states. The node behavior does, however, normalize over time. This implies that the filter eventually converges to the desired solution in finite time. Initial conditions in the EKF will likely prove a useful parameter to tailor the initial behavior to suit proposed UGV-DTN routing algorithms.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

The success of a reliable cooperative routing protocol in UGV networks is contingent upon our ability to accurately estimate the spatial positions of UGV nodes as a function of time. The idea is that the path planning strategy will use the mobility estimation model as input to achieve cooperation between individual UGV nodes for routing of information.

In this thesis we have implemented a foundational mobility estimation algorithm that can be coupled with a cooperative communication routing algorithm to provide a basis for real time path planning in UGV-DTNs.

The algorithm that is developed is this thesis is based on an EKF technique that exploits a non-linear Gauss Markov state model to reflect node dynamics. The algorithm utilizes constant power RSSI signals transmitted from fixed position base stations. The EKF uses a Jacobian matrix, derived in Chapter III, for approximate linearization of the non-linear measurements. In this thesis, position is the only state required for measurement linearization. The algorithm works with the underlying assumption that the process noise and the measurement noise have Gaussian distributions. The EKF algorithm filters recursively, estimating the current state of the UGV node. The EKF algorithm operates recursively in time, meaning that the current state vector estimate is a function of only the estimate at the last time step. The storage of additional past information is not required, so storage resource utilization for individual UGV nodes is minimized.

In our performance evaluations, we simulated a single node traveling along a trajectory that includes abrupt maneuvers. Estimation performance is assessed with zero mean whiteness tests on the innovation sequences, RMSE of the state estimates, WSSRs on the innovations, and the PCRLB. The algorithm is shown to implement efficient mobility tracking of UGV nodes in a wireless network. We have demonstrated that the mobility estimator performs effectively and therefore can be legitimately integrated into a new cooperative routing protocol with enhanced accuracy.

A. FUTURE WORK

In this thesis, the key issue of mobility estimation and prediction in a UGV-DTN is studied. This is the first step to integrating a stochastic prediction algorithm with path planning protocols in a UGV-DTN. Further research directions should address the following important issues.

1. Combination with Routing Algorithm

The formulation of a path planning strategy using the mobility prediction model developed in this thesis is the next logical and necessary step towards increased reliability in UGV-DTN communications. The mobility prediction model will be used to provide external situational awareness (i.e., the position of a UGV node at time t) which will facilitate the development of a usable communication path planning strategy for UGV nodes within an individual cluster island. A UGV-DTN requires a uniquely formulated cooperative QoS routing approach that considers realistic resource constraints and situational awareness of terrain and surrounding environment. In that regard, a cooperative communication routing algorithm among UGV nodes coupled with the mobility prediction algorithm discussed in this thesis will be developed to ensure a real time assessment of the best next forwarding nodes in terms of resource availability. The QoS in UGV-DTNs depends on the integration of mobility and situational awareness into path planning algorithms so that the probability of connectivity between a pair of UGVs is maximized and the aggregate resource consumption is minimized. The path planning strategy will be developed to uniquely suit the constraints and parameters of a UGV-DTN ad hoc network.

2. Utilization of GPS-Enabled Anchor nodes

This measurement strategy obviates the need for BSs by allowing a subset of the mobile nodes (anchor nodes) to carry GPS sensors in addition to their built-in RSSI sensors while a subset of non-anchor nodes carry only RSSI sensors. In addition, a UAV will be allowed to carry both GPS and RSSI sensors. This scheme would allow for the superior performance associated with the use of BSs, but with some savings of the costs associated with the use of GPS sensors. This measurement scheme may allow the signal

models, modified from those provided in this paper, to be observable. The adjustment of the model to utilize GPS sensors as available can be the subject of a future study.

3. Estimation Using RBPF

The process of Rao-Blackwellization is a technique for improving particle filtering by analytically marginalizing some of the variables (linear and Gaussian) from the joint posterior distributions used in particle filtering. The linear part of the system model is then estimated by a KF, an optimal estimator, while the nonlinear part is estimated by a PF. This leads to the fact that a KF is attached to each particle. In the mobility tracking problem the positions of the mobile unit are estimated with a PF, while the speeds and accelerations with a KF [15]. This type of filter is shown in [15] to decrease the computational complexity of the PF and decrease the peak-dynamic errors during abrupt maneuvers. These behaviors are important for practical use and may prove useful in the UGV-DTN mobility problem. Mobility estimation of this nature applied to the UGV-DTN problem can be the subject of a future study.

4. Estimation Using Actual UGV-DTN node mobility data

Actual UGV-DTN mobility data are not currently available. Generating data by deploying a UGV-DTN in a suitable operating environment with RSSI and GPS sensors would allow for more realistic evaluation of the algorithm proposed in this thesis and future mobility estimation algorithms for the UGV-DTN scenario. Generation and processing of this type of data with the state estimate can be the subject of a future study.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

The appendices that follow document the MATLAB code utilized for simulation of the UGV-DTN mobility estimation and tracking problem.

A. FLOW DIAGRAM OF MATLAB FUNCTIONS

This appendix presents the flow diagram of the EKF algorithms.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%  BLOCK DIAGRAM OF PROGRAM FLOW %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                            EKF_Caller_TMB.m
%                             ^     ^     ^
%                             |     |     |
%                *PLOTS*      |     |     | *PERFORMANCE TESTS*
%            -----------------      |      ---------------------
%            |                      |                          |
%            |                      |                  Whiteness_EKF_TMB
%            |                   EKF_TMB                   WSSR_TMB
%      SS_Noise_Plot             ^ ^  ^ ^                  PCRLB_TMB
%     SS_Nonlin_Plot             | |  | |                      ^
%                                | |  | |                      |
%                                | |  | |             Outlier_Counter_TMB
%                                | |  | |
%              ------------------- |  | ------------------------
%              |            --------  -----                    |
%              |            |             |                    |
%              |            |             |                    |
%           EKF_IC     Build_HH_TMB       |           SS_Model_Build_TMB
%                      Build_hk_TMB       |              ^     ^
%                           ^             |              |     |
%                           |             |              |     |
%                           ------------- | --------------     |
%                                       | | |                  |
%                                   SS_Model_Const       MC_Input_Build
%                                         ^               ^    ^
%                                         |               |    |
%                                         |  --------------    |
%                                         |  |                 |
%                                   Sim_Parameters     getTransitionMatrix
%                                                              ^
%                                                              |
%                                                              |
%                                                        getMarkovChain
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B. EKF CALLER FUNCTION

This appendix presents the function for building the Jacobian for the AGV mobile node problem.

```
        function [HH] = Build_HH_TMB(Xp)
%
%         FUNCTION: Build_HH_TMB.m
%
%         PURPOSE: Function for building the Jacobian for the AGV mobile node problem.
```

```
%
%         SOURCE:                        Matlab M-file
%         VERSION:                       1.0
%         ORIGINATION DATE:              November 28, 2012
%         DATE OF LAST MODIFICATION:     November 28, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
%
%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
               % HH = Jacobian
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  EKF_TMB.m                      % Code for EKF

%         CODES CALLED BY THIS FUNCTION:
%            1.  Sim_Parameters.m               % Passes simulation parameters

%         VARIABLES USED IN THE CODE:
%            1.  BS                             % matrix of base station coordinates in x
and y
%            2.  NBS                            % total number of base stations
%            3.  Xp                             % predicted state for current EKF
iteration
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Define simulation parameters

            [Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax]=Sim_Parameters;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % CONSTRUCT THE JACOBIAN (HH) Necessary for the EKF:

            if NBS <3
                display('Need more Base Stations for triangulation. (Sim_Parameters.m)');
                return;
            end

            if NBS > 1
                HH = [-10*eta*(Xp(1)-BS(1,1))/(log(10)*((Xp(1)-BS(1,1))^2+(Xp(4)-
BS(1,2))^2)),0,0,-10*eta*(Xp(4)-BS(1,2))/(log(10)*((Xp(1)-BS(1,1))^2+(Xp(4)-
BS(1,2))^2)),0,0;
                      -10*eta*(Xp(1)-BS(2,1))/(log(10)*((Xp(1)-BS(2,1))^2+(Xp(4)-
BS(2,2))^2)),0,0,-10*eta*(Xp(4)-BS(2,2))/(log(10)*((Xp(1)-BS(2,1))^2+(Xp(4)-
BS(2,2))^2)),0,0;
                      -10*eta*(Xp(1)-BS(3,1))/(log(10)*((Xp(1)-BS(3,1))^2+(Xp(4)-
BS(3,2))^2)),0,0,-10*eta*(Xp(4)-BS(3,2))/(log(10)*((Xp(1)-BS(3,1))^2+(Xp(4)-
BS(3,2))^2)),0,0];
            end

            if NBS > 3
                HH = [HH;-10*eta*(Xp(1)-BS(4,1))/(log(10)*((Xp(1)-BS(4,1))^2+(Xp(4)-
BS(4,2))^2)),0,0,-10*eta*(Xp(4)-BS(4,2))/(log(10)*((Xp(1)-BS(4,1))^2+(Xp(4)-
BS(4,2))^2)),0,0];
            end

            if NBS > 4
                HH = [HH;-10*eta*(Xp(1)-BS(5,1))/(log(10)*((Xp(1)-BS(5,1))^2+(Xp(4)-
BS(5,2))^2)),0,0,-10*eta*(Xp(4)-BS(5,2))/(log(10)*((Xp(1)-BS(5,1))^2+(Xp(4)-
BS(5,2))^2)),0,0];
            end
```

```
            if NBS > 5
                HH = [HH;-10*eta*(Xp(1)-BS(6,1))/(log(10)*((Xp(1)-BS(6,1))^2+(Xp(4)-
BS(6,2))^2)),0,0,-10*eta*(Xp(4)-BS(6,2))/(log(10)*((Xp(1)-BS(6,1))^2+(Xp(4)-
BS(6,2))^2)),0,0];
            end

            if NBS > 6
                HH = [HH;-10*eta*(Xp(1)-BS(7,1))/(log(10)*((Xp(1)-BS(7,1))^2+(Xp(4)-
BS(7,2))^2)),0,0,-10*eta*(Xp(4)-BS(7,2))/(log(10)*((Xp(1)-BS(7,1))^2+(Xp(4)-
BS(7,2))^2)),0,0];
            end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C. WSSR FUNCTION

This appendix presents the function that performs the WSSR test on the innovations vector.

```
           function WSSR_TMB(E,RRe,Nsamples,Nw,Nz,Ts)


%      PURPOSE:  Perform Weighted Sum Squared Residual (WSSR) test on an innovations
vector
%
%      SOURCE:                    Matlab M-files
%      VERSION:                   2.0
%      DATE FIRST WRITTEN:        December 5, 2012
%      DATE LAST MODIFIED:        December 5, 2012

%      AUTHOR:      Timothy M. Beach (TMB)
%
%      INPUTS:
%               1. E = the innovations vector (Nz by Nsamps)
%               2. RRe = Innovations covariance matrix for all times (Nz by Nz by Nsamps)
%               3. Nsamples = the number of time samples over which to compute the
variables
%               4. Nw = Window length over which to compute the WSSR
%               5. Nz = Number of output measurements in the state-space model
%               6. Ts = the temporal sampling interval
%               7. Tstart = the starting time (sample) to use for the innovations signals

%     OUTPUTS:  WSSR plot
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up some constants, etc.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    tplot_start     = Nw*Ts;                    % Time at which to start plotting WSSR
    tplot_end       = (Nsamples-1)*Ts;          % Ending time on the plot


    Cinterval = 1.96;       % The constant for the confidence interval
    alpha = .05;            % The significance level for the hypothesis test
    Nlags = Nsamples-Nw;    % The number of lags over which WSSR is calculated


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute quantities needed to plot the WSSR results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------------------------------------------------------------------------------------
    % Set up a loop for calculating the WSSR
%---------------------------------------------------------------------------------------

        WSSR = zeros(1,Nlags);  % Initialize the WSSR vector
```

```
        for m = Nw:(Nlags)
            k = m - Nw + 1;              % Define the index over which to sum the WSSR

            WSSR(m)     = WSSR(m) + E(:,k)'*(inv(RRe(:,:,k)))*E(:,k); % Sum up the
quadratic forms computing inverse each time

        end;


%-------------------------------------------------------------------------------------
    % Compute bounds (threshold(s)) and count number of samples that exceed the bounds
%-------------------------------------------------------------------------------------

        tau = Nw*Nz + 1.96*sqrt(2*Nw*Nz);          % Threshold for the WSSR test

        icount = 0;                                % Initialize the counter

        [icount,nn] = size(find(WSSR(:) > tau));    % Search for the indices of the WSSR
values that exceed the threshold, tau

        percent = (icount/Nlags)*100.;             % Find percentage of WSSR values
                                                   % that lie outside the bounds
%-------------------------------------------------------------------------------------
    % Test if more than alpha percent of the WSSR values exceed the threshold, tau
%-------------------------------------------------------------------------------------

        if (percent > alpha)               % Test if the percentage of WSSR values that
                                           % lie outside the bounds is greater than
alpha
                disp(' ')
                disp('   ##### WSSR > Tau, so the EKF is NOT tuned #####') % Print to
command window
                disp(' ')
                badcnt=1;
                good='n';

           else
                disp(' ')
                disp('   ***** WSSR < Tau, so the EKF is tuned *****') % Print to command
window
                disp(' ')
                good='y';

        end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the WSSR results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        tplot = (0:Nlags-1)*Ts + tplot_start;               % Create a time vector for
plotting WSSR

        wtitle = sprintf('Lags  (Percent  /  # samples out of bounds = %4.2f  (%g))',
percent,  icount);  % Prepare a plot title

        ttitle = sprintf('WSSR  =  Weighted Sum Squared Residuals,   Nw = %4.0f ',Nw);


        figure
        plot(tplot,WSSR,'-b',tplot,tau,'--r')
        grid on
        %title(ttitle)
      % axis([tplot_start tplot_end 0 400])           % Force my own axis limits
        ylabel('WSSR')
        xlabel(wtitle)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%   END OF M-FILE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D. WHITENESS FUNCTION

This appendix presents the function that performs a statistical whiteness test on a given signal.

```
        function Whiteness_EKF_TMB(E,Ts,Tstart)

%         PURPOSE: Perform statistical whiteness tests on a given signal
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             December 1, 2012
%         DATE OF LAST MODIFICATION:    December 1, 2012
%
%         AUTHOR:       TMB (TMB)
%
%         INPUTS:
%               1. E = the signal to test for whiteness
%               2. t = the sampling interval
%               3. Tstart = the starting time (sample) to use for the signal E


%         OUTPUTS:      None

%   Code(s) that call this function:
%           1. EKF_Caller_TMB.m

%   Codes called by this function:
%           none

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DEFINE PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        Cinterval = 1.96;       % Cinterval = the multiplier on the "sigma" of the
                                % distribution used to define the bounds

        bndpct = 5;             % bndpct = the probability used to define the confidence
                                % interval as follows:
                                % P(Lower bound < variable < Upper bound) = 1-bncpct


        Nsamps = length(E);     % E = error signal = innovations


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% DEFINE AUTOCORRELATIONS:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

         Rinnov = xcorr(E,'coeff');     % Rinnov = the normalized autocorrelation of E
(rho)

         Nlags = length(Rinnov);        % Nlags = No. of lags over which to calculate Ree


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% PRINT OUT THE MEAN OF THE INNOVATIONS (TEST FOR ZERO MEAN INNOVATIONS)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

         innovMean = sprintf('ZERO MEAN / WHITENESS TEST: (Innov. Mean = %7.2e < 2Sigma
Bound = %7.2e) ', ...
                             abs(mean(E)), Cinterval*sqrt(Rinnov(1)/Nlags));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%
% COMPUTE THE BOUNDS, ETC. OF THE CONFIDENCE INTERVAL:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

         boundup = ( Cinterval/(sqrt(Nsamps)) )*ones(Nlags,1);      % Compute upper two-
sigma bound

         icount = 0;                                                % Initialize icount
to zero
                                                                    % icount = the number
of autocorr.
                                                                    % samples that fall
outside the
                                                                    % two sigma bounds.

         ictup = find(Rinnov(2:Nlags) > boundup(1:Nlags-1));        % Find the values of
the autocorr.
                                                                    % that exceed the
upper two-sigma bound
                                                                    % "find(X)" finds the
indices and values
                                                                    % of nonzero elements
in X.  Don't count
                                                                    % the first point in
Rinnov

         ictlow = find(Rinnov(2:Nlags) < -boundup(1:Nlags-1));      % Find the values of
the autocorr.
                                                                    % that exceed the
lower bound.  Don't
                                                                    % count the first
point in Rinnov

         [ict1,nn] = size(ictup);                                   % Find the size of
ictup
         [ict2,nn] = size(ictlow);                                  % Find the size of
ictlow


         icount = ict1 + ict2;                      % Total number of autocorrelation
values
                                                        % that fall outside the bounds

         percent = (icount/Nlags)*100.;             % Find percentage of autocorrelation
values
                                                        % that lie outside the bounds

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
% PLOT WHITENESS TEST RESULTS:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%

    figure('NumberTitle','on','Name','Whiteness Test','color',[.75 .75 .75], ...
                                      'units','norm');
    subplot(1,1,1)
    wtitle = sprintf('Lag Time in Seconds  (No. Pts. Outside Bound = %3.0f  ( Percent =
%2.1g))',icount,percent);
```

```
    %---------------------------------------------------------------------------------------
---------
    % Compute the times in seconds corresponding to the lags we wish to plot
    %---------------------------------------------------------------------------------------
---------

        Tplot = (0:Nlags-1)*Ts + Tstart;                % Compute times (in seconds) for
plotting
                                                        % This is where Tstart enters the
formulation


    %---------------------------------------------------------------------------------------
---------
    % PLOT THE AUTOCORRELATION:
    %---------------------------------------------------------------------------------------
---------

        plot(Tplot,Rinnov,'-b',Tplot,boundup, '--r' ...
                                     ,Tplot,-boundup,'--r');      % Plot the
autocorrelation and the
                                                                  % two-sigma bounds
around it
        grid on;
        ylabel('Normalized Autocorrelation of E')
        %title(innovMean)                                          % Get the title from
above
        xlabel(wtitle)                                            % Get the xlabel from
above

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
% COMPUTE THE PERCENTAGE OF SAMPLES THAT LIE OUTSIDE THE BOUNDS
%       AND DECLARE THE SIGNAL TO BE WHITE OR NON-WHITE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%


      if (percent > bndpct)                       % Test if the percentage of
autocorrelation values that
                                                  % lie outside the bounds is greater than
bndpct
            disp(' ')
            disp('    ##### Non-White #####')  % If so, declare the signal to be non-
white
            badcnt=1;
            good='n';
         else

            disp(' ')
            disp('    ***** White *****')      % If not, declare the signal to be white
            good='y';

      end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   END OF M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

E. STATE SPACE NONLINEAR FUNCTION

This appendix presents the function that plots the nonlinear state space model inputs and outputs.

```
          function SS_Nonlin_Plot(t,u,x,BS,NBS,z,Zpr,E,Rin,Xcor,Xtilda,Pcor)

%
%         PURPOSE: Plot nonlinear state space model inputs, outputs
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 18, 2012
%         DATE OF LAST MODIFICATION:    December 1, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            Input arguments are of the following form:
%                t = vector of time samples used in the simulation
%                u = vector of inputs to the state space system
%                z = vector of outputs from the state space system
%                x = vector of states of the state space system


%         OUTPUTS:      None - The results are plots

%   Code(s) that call this function:
%           1. SS_Model_Build_TMB.m

%   Codes called by this function:
%           None

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            Nsamples=size(t,1);

       % Plot the command input values (u)

            figure

            hold on;
            subplot(2,1,1)
            plot(t,u(1,:),'r')
            title('Command Input in x direction (uk)')
            ylabel('Acceleration (m/s^2)')
            xlabel('Time (s)')
            grid on
            subplot(2,1,2)
            plot(t,u(2,:),'b')
            title('Command Input in y direction (uk)')
            ylabel('Acceleration (m/s^2)')
            xlabel('Time (s)')
            grid on

        % Plot the Actual Trajectory (x), Estimated Trajectoryj, Base Stations (BS)
        % and Error of the state estimate (Xtilda), Two Sigma Bounds (Bound_Xtilda)
        % and No. of points lying outside the Two Sigma Bounds

    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % DEFINE PARAMETERS
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        Cinterval = 1.96;                          % (TMB) Cinterval = the multiplier on
the "sigma" of the distribution used to define the bounds

        bndpct = 5;                                % (TMB) bndpct = the probability used
to define the confidence interval as follows: P(Lower bound < variable < Upper bound) =
1-bncpct
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % Calculate the Two Sigma Bounds for Xtilda to use on the plot
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

            clear Bound_Xtilda;
```

```
            Bound_Xtilda = (Cinterval)*sqrt(Pcor);      % Upper (Positive) Bound

    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % COUNT THE NO. OF POINTS LYING OUTSIDE THE BOUNDS
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        % Count the number of signal points lie outside the given two sigma bounds
        % for the 2 components of the vector Xtilda:

            [icount_1,percent_1] =
Outlier_Counter_TMB(Xtilda(1,:),Bound_Xtilda(1,:),Nsamples,NBS);
            [icount_2,percent_2] =
Outlier_Counter_TMB(Xtilda(2,:),Bound_Xtilda(2,:),Nsamples,NBS);
            if NBS > 2
                [icount_3,percent_3] =
Outlier_Counter_TMB(Xtilda(3,:),Bound_Xtilda(3,:),Nsamples,NBS);
            end
            if NBS > 3
                [icount_4,percent_4] =
Outlier_Counter_TMB(Xtilda(4,:),Bound_Xtilda(4,:),Nsamples,NBS);
            end
            if NBS > 4
                [icount_5,percent_5] =
Outlier_Counter_TMB(Xtilda(5,:),Bound_Xtilda(5,:),Nsamples,NBS);
            end
            if NBS > 5
                [icount_6,percent_6] =
Outlier_Counter_TMB(Xtilda(6,:),Bound_Xtilda(6,:),Nsamples,NBS);
            end
            if NBS > 6
                [icount_7,percent_7] =
Outlier_Counter_TMB(Xtilda(7,:),Bound_Xtilda(7,:),Nsamples,NBS);
            end


    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % Plot
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            figure

            hold on;
            plot(x(1,:),x(4,:),'r')
            plot(Xcor(1,:),Xcor(4,:),'--b')
            plot (BS(1,1),BS(1,2),'bd')
            plot (BS(2,1),BS(2,2),'gd')
            if NBS > 2
                plot (BS(3,1),BS(3,2),'rd')
            end
            if NBS > 3
                plot (BS(4,1),BS(4,2),'cd')
            end
            if NBS > 4
                plot (BS(5,1),BS(5,2),'md')
            end
            if NBS > 5
                plot (BS(6,1),BS(6,2),'yd')
            end
            if NBS > 6
                plot (BS(7,1),BS(7,2),'kd')
            end
            title('Trajectories');
            xlabel('x coordinate, [m]');
            ylabel('y coordinate, [m]');
            if NBS == 2
                legend('Actual Trajectory','Estimated Trajectory','BS1','BS2')
            elseif NBS == 3
                legend('Actual Trajectory','Estimated Trajectory','BS1','BS2','BS3')
            elseif NBS == 4
```

```
                legend('Actual Trajectory','Estimated
Trajectory','BS1','BS2','BS3','BS4')
            elseif NBS == 5
                legend('Actual Trajectory','Estimated
Trajectory','BS1','BS2','BS3','BS4','BS5')
            elseif NBS == 6
                legend('Actual Trajectory','Estimated
Trajectory','BS1','BS2','BS3','BS4','BS5','BS6')
            elseif NBS == 7
                legend('Actual Trajectory','Estimated
Trajectory','BS1','BS2','BS3','BS4','BS5','BS6','BS7')
            end
            grid on


            figure

            subplot(NBS,1,1)
            hold on;
            plot(t,Xtilda(1,:),'r',t,Bound_Xtilda(1,:),'--b',t,-Bound_Xtilda(1,:),'--b')
            title('Xtilda_1 = x(t) - xhat_1(t|t) and Two Sigma Bounds vs. Time');
            ylabel('Xtilda_1(t)');
            xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  ( Percent =
%2.1g))',icount_1,percent_1));
            grid on
            if NBS > 1
                subplot(NBS,1,2)
                hold on;
                plot(t,Xtilda(2,:),'r',t,Bound_Xtilda(2,:),'--b',t,-Bound_Xtilda(2,:),'--
b')
                title('Xtilda_2 = x(t) - xhat_2(t|t) and Two Sigma Bounds vs. Time');
                ylabel('Xtilda_2(t)');
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_2,percent_2));
                grid on
            end
            if NBS > 2
                subplot(NBS,1,3)
                hold on;
                plot(t,Xtilda(3,:),'r',t,Bound_Xtilda(3,:),'--b',t,-Bound_Xtilda(3,:),'--
b')
                title('Xtilda_3 = x(t) - xhat_3(t|t) and Two Sigma Bounds vs. Time');
                ylabel('Xtilda_3(t)');
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_3,percent_3));
            end
            if NBS > 3
                subplot(NBS,1,4)
                hold on;
                plot(t,Xtilda(4,:),'r',t,Bound_Xtilda(4,:),'--b',t,-Bound_Xtilda(4,:),'--
b')
                title('Xtilda_4 = x(t) - xhat_4(t|t) and Two Sigma Bounds vs. Time');
                ylabel('Xtilda_4(t)');
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_4,percent_4));
                grid on
            end
            if NBS > 4
                subplot(NBS,1,5)
                hold on;
                plot(t,Xtilda(5,:),'r',t,Bound_Xtilda(5,:),'--b',t,-Bound_Xtilda(5,:),'--
b')
                title('Xtilda_5 = x(t) - xhat_5(t|t) and Two Sigma Bounds vs. Time');
                ylabel('Xtilda_5(t)');
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_5,percent_5));
                grid on
            end
            if NBS > 5
```

```
                subplot(NBS,1,6)
                hold on;
                plot(t,Xtilda(6,:),'r',t,Bound_Xtilda(6,:),'--b',t,-Bound_Xtilda(6,:),'--
b')
                title('Xtilda_6 = x(t) - xhat_6(t|t) and Two Sigma Bounds vs. Time');
                ylabel('Xtilda_6(t)');
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_6,percent_6));
                grid on
            end
            if NBS > 6
                subplot(NBS,1,7)
                hold on;
                plot(t,Xtilda(7,:),'r',t,Bound_Xtilda(7,:),'--b',t,-Bound_Xtilda(7,:),'--
b')
                title('Xtilda_7 = x(t) - xhat_7(t|t) and Two Sigma Bounds vs. Time');
                ylabel('Xtilda_7(t)');
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_7,percent_7));
                grid on
            end

        % Plot the measurment values (z), predicted measurment values (Zpr),
        % innovations (E), Two Sigma Bounds (Bound_E) and No. of points
        % lying outside the Two Sigma Bounds

    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % COMPUTE THE BOUNDS for E, COUNT THE NO. OF POINTS LYING OUTSIDE THE BOUNDS
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        % Calculate the Two Sigma Bounds for E to use on the plot

            Bound_E = (Cinterval)*sqrt(Rin);      % 2 X Nsamples Upper (Positive) Bound


        % Count the number of signal points lie outside the given two sigma bounds
        % for the 3 components of the innovations vector E:

            [icount_1,percent_1] = Outlier_Counter_TMB(E(1,:),Bound_E(1,:),Nsamples,NBS);
            [icount_2,percent_2] = Outlier_Counter_TMB(E(2,:),Bound_E(2,:),Nsamples,NBS);
            if NBS > 2
                [icount_3,percent_3] =
Outlier_Counter_TMB(E(3,:),Bound_E(3,:),Nsamples,NBS);
            end
            if NBS > 3
                [icount_4,percent_4] =
Outlier_Counter_TMB(E(4,:),Bound_E(4,:),Nsamples,NBS);
            end
            if NBS > 4
                [icount_5,percent_5] =
Outlier_Counter_TMB(E(5,:),Bound_E(5,:),Nsamples,NBS);
            end
            if NBS > 5
                [icount_6,percent_6] =
Outlier_Counter_TMB(E(6,:),Bound_E(6,:),Nsamples,NBS);
            end
            if NBS > 6
                [icount_7,percent_7] =
Outlier_Counter_TMB(E(7,:),Bound_E(7,:),Nsamples,NBS);
            end

    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % Plot
    %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

            figure

            hold on;
            subplot(NBS,1,1)
            plot(t,z(1,:),'r',t,Zpr(1,:),'b')
```

```
title('Actual Measurement (zk_1)')
ylabel('Distance from BS_1 (m)')
xlabel('Time (s)')
grid on
subplot(NBS,1,2)
plot(t,z(2,:),'r',t,Zpr(2,:),'b')
title('Actual Measurement (zk_2)')
ylabel('Distance from BS_2 (m)')
xlabel('Time (s)')
grid on

if NBS > 2
    subplot(NBS,1,3)
    plot(t,z(3,:),'r',t,Zpr(3,:),'b')
    title('Actual Measurement (zk_3)')
    ylabel('Distance from BS_3 (m)')
    xlabel('Time (s)')
    grid on
end

elseif NBS > 3
    subplot(NBS,1,4)
    plot(t,z(4,:),'r',t,Zpr(4,:),'b')
    title('Actual Measurement (zk_4)')
    ylabel('Distance from BS_4 (m)')
    xlabel('Time (s)')
    grid on
end

elseif NBS > 4
    subplot(NBS,1,5)
    plot(t,z(5,:),'r',t,Zpr(5,:),'b')
    title('Actual Measurement (zk_5)')
    ylabel('Distance from BS_5 (m)')
    xlabel('Time (s)')
    grid on
end

if NBS > 5
    subplot(NBS,1,6)
    plot(t,z(6,:),'r',t,Zpr(6,:),'b')
    title('Actual Measurement (zk_6)')
    ylabel('Distance from BS_6 (m)')
    xlabel('Time (s)')
    grid on
end

if NBS > 6
    subplot(NBS,1,7)
    plot(t,z(7,:),'r',t,Zpr(7,:),'b')
    title('Actual Measurement (zk_7)')
    ylabel('Distance from BS_7 (m)')
    xlabel('Time (s)')
    grid on
end


figure

hold on
subplot(NBS,1,1)
plot(t,E(1,:),'-r',t,Bound_E(1,:),'--b',t,-Bound_E(1,:),'--b');
title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  ( Percent =
%2.1g))',icount_1,percent_1))
ylabel('E_1(t)')
grid on
subplot(2,1,2)
plot(t,E(2,:),'-r',t,Bound_E(2,:),'--b',t,-Bound_E(2,:),'--b');
```

```
            title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
            xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  ( Percent =
%2.1g))',icount_2,percent_2))
            ylabel('E_2(t)')
            grid on



          if NBS > 2
               figure

               hold on;
               subplot(2,1,1)
               plot(t,z(3,:),'r',t,Zpr(3,:),'b')
               title('Actual Measurement (zk_3)')
               ylabel('Distance from BS_3 (m)')
               xlabel('Time (s)')
               grid on
               subplot(2,1,2)
               plot(t,E(3,:),'-r',t,Bound_E(3,:),'--b',t,-Bound_E(3,:),'--b');
               title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
               xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_3,percent_3))
               ylabel('E_3(t)')
               grid on
            end

            if NBS > 3
               figure

               hold on;
               subplot(2,1,1)
               plot(t,z(4,:),'r',t,Zpr(4,:),'b')
               title('Actual Measurement (zk_4)')
               ylabel('Distance from BS_4 (m)')
               xlabel('Time (s)')
               grid on
               subplot(2,1,2)
               plot(t,E(4,:),'-r',t,Bound_E(4,:),'--b',t,-Bound_E(4,:),'--b');
               title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
               xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_4,percent_4))
               ylabel('E_4(t)')
               grid on
            end

            if NBS > 4
               figure

               hold on;
               subplot(2,1,1)
               plot(t,z(5,:),'r',t,Zpr(5,:),'b')
               title('Actual Measurement (zk_5)')
               ylabel('Distance from BS_5 (m)')
               xlabel('Time (s)')
               grid on
               subplot(2,1,2)
               plot(t,E(5,:),'-r',t,Bound_E(5,:),'--b',t,-Bound_E(5,:),'--b');
               title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
               xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_5,percent_5))
               ylabel('E_5(t)')
               grid on
            end

            if NBS > 5
               figure
```

```
                hold on;
                subplot(2,1,1)
                plot(t,z(6,:),'r',t,Zpr(6,:),'b')
                title('Actual Measurement (zk_6)')
                ylabel('Distance from BS_6 (m)')
                xlabel('Time (s)')
                grid on
                subplot(2,1,2)
                plot(t,E(6,:),'-r',t,Bound_E(6,:),'--b',t,-Bound_E(6,:),'--b');
                title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_6,percent_6))
                ylabel('E_6(t)')
                grid on
            end

            if NBS > 6
                figure

                hold on;
                subplot(2,1,1)
                plot(t,z(7,:),'r',t,Zpr(7,:),'b')
                title('Actual Measurement (zk_7)')
                ylabel('Distance from BS_7 (m)')
                xlabel('Time (s)')
                grid on
                subplot(2,1,2)
                plot(t,E(7,:),'-r',t,Bound_E(7,:),'--b',t,-Bound_E(7,:),'--b');
                title('Innovations E = e(t) and Two Sigma Bounds vs. Time')
                xlabel(sprintf('Time in Seconds  (No. Pts. Outside Bound = %3.0g  (
Percent = %2.1g))',icount_7,percent_7))
                ylabel('E_7(t)')
                grid on
            end


 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  End of M-File   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

F. STATE SPACE NOISE FUNCTION

This appendix presents the function that plots the process and measurement noise of the state space model.

```
        function SS_Noise_Plot(NBS,t,V,W)

%
%         FUNCTION: SS_Noinse_Plot
%
%         PURPOSE: Plot process and measurment noise of the SS model
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 18, 2012
%         DATE OF LAST MODIFICATION:    November 18, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            Input arguments are of the following form:
%                t = vector of time samples used in the simulation
%                u = vector of inputs to the state space system
%                z = vector of outputs from the state space system
%                x = vector of states of the state space system


%         OUTPUTS:      None - The results are plots
```

```
%   Code(s) that call this function:
%           1. SS_Model_Build_TMB.m

%   Codes called by this function:
%           None

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    [Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax,p_t]=Sim_Parameters;

% Condition Matrices for plotting

    V=V.';
    W=W.';

% Plot the SS model noise sequences
    figure

    figname=['Process Noise W(t):   '];
    subplot(2,1,1)
    plot(t,W(:,1),'b',t,2*stdW*(ones(size(t))),'--r',t,-2*stdW*(ones(size(t))),'--r')
    %title('Process Noise W_x(t) vs. Time');
    xlabel('Time in Seconds');
    ylabel('W_x(t)');
    grid on
    subplot(2,1,2)
    plot(t,W(:,2),'r',t,2*stdW*(ones(size(t))),'--r',t,-2*stdW*(ones(size(t))),'--r')
    %title('Process Noise W_y(t) vs. Time');
    xlabel('Time in Seconds');
    ylabel('W_y(t)');
    grid on

    figure

    figname=['Measurement Noise V(t):   '];
    subplot(NBS,1,1)
    plot(t,V(:,1),'b',t,2*stdV*(ones(size(t))),'--r',t,-2*stdV*(ones(size(t))),'--r')
    %title('Measurement Noise V_1(t) vs. Time');
    xlabel('Time in Seconds');
    ylabel('V_1(t)');
    grid on
    subplot(NBS,1,2)
    plot(t,V(:,2),'r',t,2*stdV*(ones(size(t))),'--b',t,-2*stdV*(ones(size(t))),'--b')
    %title('Measurement Noise V_2(t) vs. Time');
    xlabel('Time in Seconds');
    ylabel('V_2(t)');
    if (NBS > 2)
        subplot(NBS,1,3)
        plot(t,V(:,3),'g',t,2*stdV*(ones(size(t))),'--b',t,-2*stdV*(ones(size(t))),'--b')
        %title('Measurement Noise V_3(t) vs. Time');
        xlabel('Time in Seconds');
        ylabel('V_3(t)');
    end
    if (NBS > 3)
        subplot(NBS,1,4)
        plot(t,V(:,4),'c',t,2*stdV*(ones(size(t))),'--b',t,-2*stdV*(ones(size(t))),'--b')
        title('Measurement Noise V_4(t) vs. Time');
        xlabel('Time in Seconds');
        ylabel('V_4(t)');
    end
    if (NBS > 4)
        subplot(NBS,1,5)
        plot(t,V(:,5),'m',t,2*stdV*(ones(size(t))),'--b',t,-2*stdV*(ones(size(t))),'--b')
        title('Measurement Noise V_5(t) vs. Time');
        xlabel('Time in Seconds');
        ylabel('V_5(t)');
    end
    if (NBS > 5)
```

```
        subplot(NBS,1,6)
        plot(t,V(:,6),'y',t,2*stdV*(ones(size(t))),'--b',t,-2*stdV*(ones(size(t))),'--b')
        title('Measurement Noise V_6(t) vs. Time');
        xlabel('Time in Seconds');
        ylabel('V_6(t)');
    end
    if (NBS > 6)
        subplot(NBS,1,7)
        plot(t,V(:,7),'k',t,2*stdV*(ones(size(t))),'--b',t,-2*stdV*(ones(size(t))),'--b')
        title('Measurement Noise V_7(t) vs. Time');
        xlabel('Time in Seconds');
        ylabel('V_7(t)');
    end
    grid on


% Plot the noise statistics for diagnostic purposes

     figure

     figname=['Histogram of the Process Noise W(t):   '];
     hist(W);    % Plot the Histogram
     %title('Histogram of the Process Noise W(t)');
     xlabel('Bin');
     ylabel('Number of Counts in Each Bin');
     legend('W_1(t)','W_2(t)')
     grid on

     figure

     figname=['Histogram of the Measurement Noise V(t):   '];
     hist(V);    % Plot the Histogram
     %title('Histogram of the Measurement Noise V(t)');
     xlabel('Bin');
     ylabel('Number of Counts in Each Bin');
     if NBS == 2
         legend('V_1(t)','V_2(t)')
     end
     if NBS == 3
          legend('V_1(t)','V_2(t)','V_3(t)')
     end
     if NBS == 4
          legend('V_1(t)','V_2(t)','V_3(t)','V_4(t)')
     end
     if NBS == 5
          legend('V_1(t)','V_2(t)','V_3(t)','V_4(t)','V_5(t)')
     end
     if NBS == 6
          legend('V_1(t)','V_2(t)','V_3(t)','V_4(t)','V_5(t)','V_6(t)')
     end
     if NBS == 7
          legend('V_1(t)','V_2(t)','V_3(t)','V_4(t)','V_5(t)','V_6(t)','V_7(t)')
     end
     grid on


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  End of M-File   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

G. STATE SPACE MODEL CONSTANT FUNCTION

This appendix presents the function that defines the state space model constants.

```
function [a,bu,bw]=SS_Model_Const(Ts,alpha)

%
%         FUNCTION: SS_Model_Const.m
%
%         PURPOSE: Function for defining the State Space Model Constants.
```

```
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 18, 2012
%         DATE OF LAST MODIFICATION:    November 18, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
               % Ts = Sampling period for the discrete-time signals
               % alpha = reciprocal of the maneuvering constant
%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
               % a = the discrete system matrix (Nx by Nx)
               % bu = the discrete input matrix (Nx by Nu)
               % bw = the discrete process noise matrix (Nx by Nu)
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  SS_Model_Build_TMB.m          % Gauss_Markov model builder code
%            2.  EKF_TMB.m                     % Code for EKF

%         CODES CALLED BY THIS FUNCTION:
%            None

%         VARIABLES USED IN THE CODE:
%            None
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %Define constants used in the State Space Model builder

                a = [1, Ts, (Ts^2)/2, 0,  0,        0;      % Discrete system matrix
                     0,  1,       Ts, 0,  0,        0;
                     0,  0,    alpha, 0,  0,        0;
                     0,  0,        0, 1, Ts, (Ts^2)/2;
                     0,  0,        0, 0,  1,       Ts;
                     0,  0,        0, 0,  0,    alpha];
                bu = [(Ts^2)/2,        0;                   % Discrete input matrix
                            Ts,        0;
                             0,        0;
                             0, (Ts^2)/2;
                             0,       Ts;
                             0,        0];
                bw = [(Ts^2)/2,        0;                   % Discrete process noise
matrix
                            Ts,        0;
                             1,        0;
                             0, (Ts^2)/2;
                             0,       Ts;
                             0,        1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%   END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

H. STATE SPACE MODEL BUILD FUNCTION

This appendix presents the function that builds the mobile node discrete-time linear and nonlinear ODE models for the AGV mobile node problem.

```
        function [h,W,V,x,z] = SS_Model_Build_TMB(u)
%
%         FUNCTION: SS_Model_Build_TMB.m
```

```
%
%         PURPOSE: Function for building the Mobile node discrete-time linear and
nonlinear
%                  ODE models for the AGV mobile node problem.
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 4, 2012
%         DATE OF LAST MODIFICATION:    December 1, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
               % Ts = Sampling period for the discrete-time signals
               % t = a vector of time values of size Nsamples by 1
               % Nsamples = size of vector t or number of time intervals TS
               % stdW = Standard Deviation of the process noise wk
               % stdV = Standard Deviation of the measurement noise vk
               % BS = Base Station location matrix
               % uk = input matrix
%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
               % h = the discrete measurment matrix (Nz by 1)
               % W = the discrete process noise (Nu by 1)
               % V = the discrete measurement noise (Nu by 1)
               % x = actual state vector
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  EKF_Caller_TMB.m              % Supervisor code for the EKF

%         CODES CALLED BY THIS FUNCTION:
%            1.  Sim_Parameters.m              % Passes simulation parameters
%            2.  SS_Model_Const.m              % Passes state space model constants

%         VARIABLES USED IN THE CODE:
%            1.  MeanW                         % Mean of discrete process noise
%            2.  sigmaW                        % Standard deviation of discrete process
noise
%            3.  varianceW                     % Variance of discrete process noise
%            4.  MeanV                         % Mean of discrete measurement noise
%            5.  sigmaV                        % Standard deviation of discrete
measurement noise
%            6.  varianceV                     % Variation of discrete measurement noise
%            7.  Nx                            % Number of states
%            8.  hi                            % Single node measurment calculation
%            9.  dk                            % Single distance between the node and the
BS used for node measurement
%            10. k                             % Used as increment for loop building SS
matrices (x and h)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Define simulation parameters


[Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax,p_t]=Sim_Parameters;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % BUILD THE NONLINEAR STATE-SPACE MODEL (ODE's) for the AGV Node


    % Build a random number generator to generate a zero mean white Gaussian noise
        % sequence W ~ N[0,covW] to simulate the process noise
        % sequence V ~ N[0,covV] to simulate the measurement noise
```

```
        % Fill the vector W with noise N[0,stdW] and V with noise N[0,stdV]
                % I used the "random" function as follows:
                %   W = random(NAME,A,B,M,N) & V = random(NAME,A,B,M,N), where:
                %       NAME = name of the distribution
                %       A = mean desired
                %       B = standard deviation desired
                %       M,N = size of the array I wish to generate

                W = random('Normal',0,stdW,size(t,1),2)';       % AWGN for process

                V = random('Normal',0,stdV,size(t,1),NBS)';     % AWGN for measurement


            % Compute statistics of the noises for diagnostic purposes:

                    MeanW = mean(W);
                    sigmaW = std(W);
                    varianceW = sigmaW.^2;
                    MeanV = mean(V);
                    sigmaV = std(V);
                    varianceV = sigmaV.^2;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Loop to evaluate the dynamic difference equations

        % Define constants in the model, so I don't have to calculate
        %    them at each index


            [a,bu,bw]=SS_Model_Const(Ts,alpha);


        % Initial condition for the state matrix

            x=x0;

        % Initial condition for measurement matrix

             h=[];


        % Loops:
             for k = 1:Nsamples-1

%                     size(u(:,k))
%                     size(W(:,k))
%                     size(x(:,k))
                    x = [x,a*x(:,k) + bu*u(:,k+1)];% + bw*W(:,k+1)];        % State
equation (Linear)

                    if abs(x(2,k+1)) > vmax
                        x(2,k+1) = sign(x(2,k+1))*vmax;
                    end
                    if abs(x(5,k+1)) > vmax
                        x(5,k+1) = sign(x(5,k+1))*vmax;
                    end

                    hi=[];


                    for j = 1:NBS                                   % Measurement
equation (Nonlinear)

                        dk = sqrt(((x(1,k))-BS(j,1)).^2+((x(4,k)-BS(j,2)).^2));
                        hi = [hi,z0(j) - 10*eta*log10(dk)];

                    end
```

```
                    h=[h;hi];

                end


            % Correct the "size" of the vectors so they match the convention I defined for W,
V, etc.

                    hi=[];

                    for j = 1:NBS                                       % Measurement equation
(Nonlinear)

                        dk = sqrt(((x(1,Nsamples))-BS(j,1)).^2+((x(4,Nsamples)-BS(j,2)).^2));
                        hi = [hi,z0(j) - 10*eta*log10(dk)];

                    end

                    h=[h;hi];

                    h=h.';
                    z=h+V;       % Measurements for plotting


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

I. SIMULATION PARAMETERS FUNCTION

This appendix presents the function that defines the simulation parameters for the AGV mobile node problem.

```
        function
[Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax,p_t]=Sim_Parameters

%
%         FUNCTION: Sim_Parameters.m
%
%         PURPOSE: Function for defining the Simulation Parameters.
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 18, 2012
%         DATE OF LAST MODIFICATION:    November 18, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
               % None

%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
               % BS = matrix of base station coordinates in x and y
               % NBS = total number of base station
               % x0 = initial position of AGV in x and y
               % Ts = discretisation time step
               % alpha = reciprocal of the maneuvering constant or correlation
coefficient
               % eta = slope index constant
               % z0 = base station transmission power
               % stdW = covariance of the process noise W
               % stdV = covariance of the process noise V
               % vmax = maximum AGV speed
```

```
                % p_t = transition probability p between states
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  EKF_Caller_TMB.m                 % Supervisor code for EKF
%            2.  MC_Input_Build_TMB.m             % Markov Chain input builder code
%            3.  SS_Model_Build_TMB.m             % Gauss_Markov model builder code
%            4.  EKF_TMB.m                        % Code for EKF
%            5.  Build_P_TMB.m                    % Code for building PCRLB covariance
matrix

%         CODES CALLED BY THIS FUNCTION:
%            None

%         VARIABLES USED IN THE CODE:
%            1.  Tstart = simulation start time
%            2.  Tfinal = simulation stop time
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Define the Gauss Markov model order (0 is default test, 1 is AGV node)
                Mord=0;
                %Mord=1;

        % Define Base Station (BS) and Number of Base Stations (NBS)

                BS = [4000,  9700;
                      7000, 11400;
                      6000,  9000];
                NBS = size(BS,1);

        % Define initial state of AGV node in x and y

                x0 = [3500;10;0;8500;10;0];

        % Define constants used in the State Space Model builder

                Ts = 0.5;
                Tstart = 0;
                Tfinal = 300;
                t = (Tstart:Ts:Tfinal)';
                Nsamples = ((Tfinal-Tstart)/Ts) +1;
                alpha = 0.6;
                eta = 3;
                z0 = 90*ones(NBS,1);
                stdW = 0.5;
                stdV = 4;
                vmax = 45;
                p_t = 0.8;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

J. OUTLIER COUNTER FUNCTION

This appendix presents the function that counts the number of signal points that lie outside the give two sigma bounds of a given sequence.

```
            function [icount,percent] = Outlier_Counter_TMB(E,Bound_E,Nsamples,NBS)
            % Count the number of signal points lie outside the given two sigma bounds


%   CODE NAME:      Outlier_Counter_TMB.m
%                   Count the number of signal points that lie outside the given two
sigma bounds
%
```

```
%
%         SOURCE:   Outlier_Counter_TMB.m

%         PURPOSE: Given a particular waveform and its 2*sigma bounds,
%                  this code counts the number of points that lie outside the bounds
%
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 29, 2012
%         DATE OF LAST MODIFICATION:    December 5, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:

                %   E           = 1 X Nsamples signal/waveform that is being plotted and
analyzed
                %   Bound_E     = 1 X Nsamples (scalar) two sigma bound on the signal E

%         OUTPUTS:      Later

%           Code(s) that call this function:
%               1. SS_Nonlin_Plot.m

%           Codes called by this function:
%               none


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INITIALIZE VARIABLES BEFORE WE START:

        icount  = 0;
        ictup   = 0;
        ictlow  = 0;
        ict1    = 0;
        ict2    = 0;
        nn      = 0;
        percent = 0;

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COMPUTE THE BOUNDS for E, COUNT THE NO. OF POINTS LYING OUTSIDE THE BOUNDS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Change a variable name, so existing code can use it easily:

         boundup = Bound_E;                                         % Variable name
change - Bound_E is 1 X Nsamples


    % COUNT:

         icount = 0;                                                % Initialize icount
to zero
                                                                    % icount = the number
of signal
                                                                    % samples that fall
outside the
                                                                    % two sigma bounds.

         ictup = find(E(2:Nsamples) > boundup(1:Nsamples-1));       % Find the E values
                                                                    % that exceed the
upper two-sigma bound
                                                                    % "find(X)" finds the
indices and values
                                                                    % of nonzero elements
in X.  Don't count
```

```
                                                                  % the first point in
E.

         ictlow = find(E(2:Nsamples) < -boundup(1:Nsamples-1));     % Find the values of
the innovations
                                                                  % that exceed the
lower bound.  Don't
                                                                  % count the first
point in E

         [ict1,nn] = size(ictup);                                 % Find the size of
ictup
         [ict2,nn] = size(ictlow);                                % Find the size of
ictlow


         icount = ict1 + ict2;                                    % Total number of
innovation values
                                                                  % that fall outside
the bounds

         percent = (icount/(Nsamples-1))*100.;                    % Find percentage of
innovation values
                                                                  % that lie outside
the boun

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% End of M-File %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

K. MARKOV CHAIN INPUT BUILDER FUNCTION

This appendix presents the function that builds the Markov Chain input model for the AGV mobility problem.

```
    function  [u] = MC_Input_Build_TMB
%
%
%         FUNCTION: MC_Input_Build_TMB.m

%         PURPOSE: Function for building the Markov Chain input model for the AGV
mobility problem
%
%         SOURCE:                         Matlab M-file
%         VERSION:                        1.0
%         ORIGINATION DATE:               November 7, 2012
%         DATE OF LAST MODIFICATION:      December 5, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
               % None

%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
               % Nx = the number of states
               % Nz = the number of measurements
               % W = the discrete process noise (Nu by 1)
               % h = the discrete measurment matrix (Nz by 1)
               % V = the discrete measurement noise (Nu by 1)
               % u = random acceleration command input vector (Nsamples by 1)
               % Rw = system or process noise matrices (Nx x Nx)
               % Rv = measurement noise  matrices (Nz x Nz)
```

```
%         CODES THAT CALL THIS FUNCTION:
%            1.  Sim_Supervisor_TMB.m            % Supervisory code for the simulator
%            2.  EKF_Caller_TMB.m                % Code to call the Extended Kalman
Filter


%         CODES CALLED BY THIS FUNCTION:
%            1.  Sim_Parameters                  % Passes the simulation parameters
%            2.  getMarkovChain.m                % Constructs the n order Markov Chain
%            3.  getTransitionMatrix.m           % Constructs the n order Markov Chain
Transition Matrix

%         VARIABLES USED IN THE CODE:
%            1.  Axmax                           % Maximum acceleration in x direction
%            2.  Aymax                           % Maximum acceleration in y direction
%            3.  Mx                              % Set of discrete acceleration levels in
x direction
%            4.  My                              % Set of discrete acceleration levels in
y direction
%            5.  k                               % Increment for loop building command
input matrix
%            6.  M                               % Markov Chain set returned by
getMarkovChain.m
%            7.  p                               % Markov Chain Transition Matrix
returned by getTransitionMatrix.m
%            8.  pij                             % Markov Chain Transition Matrix of all
time storing p


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ask the user to enter desired parameters (or just hard-wire them):

    % Define simulation parameters


[Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax,p_t]=Sim_Parameters;


%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::
% Define the discrete acceleration levels

            Axmax = 5;        %Maximum acceleration in x direction [m/s^2]
            Aymax = 5;        %Maximum acceleration in y direction [m/s^2]
            Mx = [-Axmax,-Axmax/3.0,-Axmax/30.0,1,Axmax/30.0,Axmax/3.0,Axmax];
            My = [-Aymax,-Aymax/3.0,-Aymax/30.0,1,Aymax/30.0,Aymax/3.0,Aymax];

% Run the code written to create the discrete-time Markov-Chain input

            if Mord==0 %Zero order is the deterministic track generator designed by user
%               u=zeros(Nsamples,2);
                u1=zeros(Nsamples/(Nsamples/300)+1,2);
                u2=[3.5*ones(15,1),zeros(15,1)];
                u3=zeros(Nsamples/(Nsamples/300)-215,2);
                u4=[-3.5*ones(25,1),zeros(25,1)];
                u5=[zeros(3,1),3.5*ones(3,1)];
                total=Nsamples-(Nsamples/(Nsamples/300)+1)-15-(Nsamples/(Nsamples/300)-
215)-28;
                u6=zeros(total,2);
                u=[u1;u2;u3;u4;u5;u6];
            else
                for k = 1:Nsamples

                    [M] = getMarkovChain(Mord,Mx,My);
                    [p] = getTransitionMatrix(M);
                    pij(k)=p;
                    u(k,1)=M(p,1);
                    u(k,2)=M(p,2);
```

```
        end

    end

    u=u.';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END OF M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

L. GET TRANSITION MATRIX FUNCTION

This appendix presents the function that constructs the transition matrix of a first order Markov chain.

```
function [transitionMatrix] = getTransitionMatrix(markovChain)
%%
% function getTransitionMatrix constructs the transition matrix of
% a first order markov chain, given the markovChain.
%
% Inputs:
%    markovChain : the markov chain, in integers.
% Ouputs:
%    transitionMatrix: the state-transition matrix (TM), where each value represents
%                      the number of occurrence for a sequence of states, which is the
%                      previous state (column of TM) followed by the current state (row
of TM).
%                      (See references for more info.)
%    Nstates: the number of states in Markov Chain IOT plot.
%
%%
% ref: http://en.wikipedia.org/wiki/Markov_chain
%      http://stackoverflow.com/questions/11072206/constructing-a-multi-order-markov-
chain-transition-matrix-in-matlab
%
% $ version 1   $ by TMB $ 31OCT2012 $ created for command input u(t) generation $
%
%%
Norder=1;

if nargin < 1,
    display('Need more data for the 1st input. (getTransitionMatrix.m)');
    return;
end

if numel(markovChain) <= Norder
    display('Need more data for the 1st input. (getTransitionMatrix.m)');
    return;
end

%make markovChain a column
if size(markovChain,1) > 1;
    markovChain = markovChain';
end

%number of states
Nstates = size(markovChain,2);

%get transition matrix
%if(Mord==1)
    transitionMatrix =  fix(Nstates*rand)+1;      % fix rounds down to nearest integer
                                                  % Nstates*rand generates integer
[0,Nstates-1]
                                                  % +1 moves values to integer
[1,Nstates]
```

```
% else
%     transitionMatrix = 0;
% end

end
```

M. GET MARKOV CHAIN FUNCTION

This appendix presents the function that constructs the first order Markov chain given the maximum acceleration levels in the x and y direction.

```
function [markovChain] = getMarkovChain(Mord,Mx,My)
%%
% function getMarkovChain constructs the first order Markov Chain
% given the acceleration levels in the x and y direction.
%
% Inputs:
%    Mord: Markov Chain Order (allow for default testing based on Ristic p.3594)
%    Mx: x direction acceleration levels.
%    My: y direction acceleration levels.
% Ouptuts:
%    markovChain : the markov chain, in integers.
%%
% %Example 1:
% %outputs the 1st order transition matrix of the below markov chain based on Ristic
p.3594.
%  uk range [-5,5] [m/s^2]
%  markovChain = M= Mx X My = markovChain=[0.0,0.0;3.5,0;0.0,3.5;0.0,-3.5;-3.5,0.0]
[m/s^2]; Norder = 1;
% [transitionMatrix,columnStates,Nstates] = getTransitionMatrix(markovChain,Norder);
%%
% ref: http://en.wikipedia.org/wiki/Markov_chain
%      http://stackoverflow.com/questions/11072206/constructing-a-multi-order-markov-
chain-transition-matrix-in-matlab
%
% $ version 1   $ by TMB $ 31OCT2012 $ created for command input u(t) generation $
%
%%

if(Mord==1)
    a=3.5;
    markovChain=[0];
else
    %make markovChain
    y=length(Mx);
    x=length(My);
    markovChain=[];

    for a=1:y
     for b=1:x
        markovChain=[markovChain;Mx(a),My(b)];
     end
    end
end


end
```

N. EKF FUNCTION

This appendix presents the function that is the supervisor code to implement the EKF within the algorithm.

```
      function [Xc,Pc,K,inov,Rinov,zp] = EKF_TMB(Nx,Rw,Rv,Pc,Xc,zk,uk)
```

```
%
%
%         SOURCE:   EKF_TMB.m

%         PURPOSE: This is a supervisor code to implement an Extended Kalman Filter (EKF)
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 4, 2012
%         DATE OF LAST MODIFICATION:    December 1, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%               a   =   Linear system or process matrix (Nx x Nx)
%               bu  =   Linear system input or command process matrix (Nx x Nu)
%               h   =   Nonlinear measurement matrix (Nz x 1)
%               HH  =   Jacobian of the nonlinear measurement matrix h (Nz x Nx)
%               Rw  =   system or process noise matrices (Nx x Nx)
%               Rv  =   measurement noise  matrices (Nx x Nx)
%               Xc  =   Corrected state vector (Nx x 1):   Xc = xhat(t-1|t-1)
%               Pc  =   Corrected state error covariance matrix (Nx x Nx): Pc = Phat(t-
1|t-1)
%               zk  =   Measurement vector (Nz x 1) at time t
%               uk  =   Input vector (Nu x 1) at time t

%
%       OUTPUTS:
%               Xc  =   Corrected state estimate vector (Nx x 1)
%               Pc  =   Corrected state error covariance matrix (Nx x Nx)
%               K   =   Kalman gain or weighting matrix (Nx x Nz)
%               inov =  Innovations sequence (residual) (Nz x 1)
%               Rinov = Innovations covariance matrix (Nz x Nz)
%               zp  =   Predicted (filtered) measurement vector (Nz x 1)
%

%   Code(s) that call this function:
%               1. EKF_Caller_TMB.m               % Supervisor code for EKF

%   Code(s) called by this function:
%               1.  Sim_Parameters.m              % Passes simulation parameters
%               2.  SS_Model_Const.m              % Passes State Space model constants
%               3.  Build_HH_TMB.m                % Build Jacobian for EKF
%               4.  Build_hk_TMB.m                % Build Measurement Prediction for EKF

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% C0NSTRUCT THE LINEAR AND NONLINEAR FUNCTIONS (a,bu) for the system model:

           [Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax,p_t] = 
Sim_Parameters;

           [a,bu,bw] = SS_Model_Const(Ts,alpha);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PREDICTION:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            Xp = a*Xc + bu*uk;                % (TMB's) state prediction for the linear 
problem
            if abs(Xp(2)) > vmax              % Xp = Xhat(t|t-1) with velocity 
limitation
                Xp(2) = sign(Xp(2))*vmax;
            end
            if abs(Xp(5)) > vmax
                Xp(5) = sign(Xp(5))*vmax;
            end
```

```
        %-------------------------------------------------------------------------

            [HH] = Build_HH_TMB(Xp);           % (TMB's) Jacobian

        %-------------------------------------------------------------------------

            [hk] = Build_hk_TMB(Xp);           % (TMB's) measurement prediction hk =
h[xhat(t|t-1)]

        %-------------------------------------------------------------------------

            Pp= a*Pc*a' + Rw;                  % (TMB's) state prediction covariance Pp =
Ptilda(t|t-1) for
                                               % the linear "stuctures" problem

        %-------------------------------------------------------------------------

            zp = hk;                            % TMB's predicted measurement for the
nonlinear problem
                                                % zp = zhat(t|t-1)

%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% INNOVATION:
%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

            inov = zk - zp;                    % (TMB's) innovation

       %-------------------------------------------------------------------------

            Rinov = HH*Pp*HH' + Rv;            % (TMB's) innovation covariance for
nonlinear problem
                                               % Note: HH is the Jacobian for measurement
equation

%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% GAIN:
%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

            K = (Pp*HH')*inv(Rinov);               % TMB's Kalman gain for the nonlinear
problem
                                               % Note: HH is the Jacobian for the
measurement equation

%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% CORRECTION:
%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

       %-------------------------------------------------------------------------

            Xc = Xp + K*inov;                  % TMB's corrected state estimate for the
linear problem
            if abs(Xc(2)) > vmax               % with velocity limitation
               Xc(2) = sign(Xc(2))*vmax;
            end
            if abs(Xc(5)) > vmax
               Xc(5) = sign(Xc(5))*vmax;
            end

       %-------------------------------------------------------------------------

            Pc = (eye(Nx)- K*HH)*Pp*(eye(Nx)- K*HH)' + K*Rv*K';   % (TMB's) corrected
covariance estimate for the  nonlinear problem. Note: HH is the Jacobian for the
measurement equation

%:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  End of M-File %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

O. EKF INITIAL CONDITIONS FUNCTION

This appendix presents the function that defines the EKF initial conditions.

```
function [Xc,Pc]=EKF_IC(Nx)

%
%         FUNCTION: EKF_IC.m
%
%         PURPOSE: Function for defining the EKF Initial Conditions Xc, PC.
%
%         SOURCE:                        Matlab M-file
%         VERSION:                       1.0
%         ORIGINATION DATE:              November 18, 2012
%         DATE OF LAST MODIFICATION:     November 18, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
                % None

%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
                % Xc = initial state matrix
                % Pc = initial covariance matrix
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  EKF_Caller_TMB.m              % Supervisor code for the EKF

%         CODES CALLED BY THIS FUNCTION:
%            None

%         VARIABLES USED IN THE CODE:
%            1.  VarX                          % Initial covariance error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Construct the initial error covariance matrix Pc
%     (subscript C is for Corrected State--just like a hat)

    % TMB's Pc for the AGV mobility problem:
            VarX = diag([400^2;15^2;5^2]);  % Initial error covariance large due to high
                                            % level of uncertainty in first
                                            % initial EKF values
            Pc = blkdiag(VarX,VarX);        % Create diagonal matrix for P0
                                            % Size of P0 = Nx by Nx

% Construct the initial state vector estimate Xc

    % TMB's Xc for the AGV mobility problem:
            Xc = [3400;5;0;8700;8;0];       % Intial estimated state input for EKF


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

P. BUILD PCRLB FUNCTION

This appendix presents the function that builds the true covariance matrix needed for PCRLB and RMSE calculations for AGV node.

```
function [P] = Build_P_TMB(Nx,Nz,x,W,h,z,V,u)
```

```
%         PURPOSE: This is a code to build the true covariance matrix
%                  needed for RMSE calculations for the AGV node problem
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             December 5, 2012
%         DATE OF LAST MODIFICATION:    December 5, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%               None

%         OUTPUTS:
%               Xc = corrected state estimate vector (Nx x 1)
%               Pc = corrected state error covariance matrix (Nx x Nx)
%               K = Kalman gain or weighting matrix (Nx x Nz)
%               inov = innovations sequence (residual) (Nz x 1)
%               Rinov = innovations covariance matrix (Nz x Nz) at time t
%               RRe = Innovations covariance matrix for all time (Nz by Nz by Nsamps)
%               zp = predicted (filtered) measurement vector (Nz x 1)

%   CODES THAT CALL THIS FUNCTION:
%               1.  State_Est_Sup_GAC.m             %

%   CODES CALLED BY THIS FUNCTION:
%               1.  Sim_Parameters.m                % Pass Simulation parameters
%               2.  MC_Input_Build_TMB.m            % Code to define the Gauss-Markov
state-space model
%               3.  EKF_TMB.m                       % EKF algorithm
%               4.  EKF_IC.m                        % EKF Initial Conditions

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% DEFINE PARAMETERS TO BE USED
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

            % Pass simulation parameters

[Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax,p_t]=Sim_Parameters;

            % Define EKF specific parameters

                    time=t;                             % time for plots
                    Nmod=fix(Nsamples/10);              % print index every Nmod

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INITIAL VARIABLES, VECTORS AND MATRICES USED FOR STATE ESTIMATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                    [Xc,Pc]=EKF_IC(Nx);                         % EKF Initial Conditions

                    [a,bu,bw] = SS_Model_Const(Ts,alpha);       % State Spacial Model
Constants

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% CONSTRUCT THE PROCESS NOISE MATRIX Rw

    % TMB's  Rw for the AGV mobility problem example problem:
                    Rw = zeros(Nx,Nx);              % Create a diagonal matrix for Rw
                    Rw(1,1)=stdW^2;Rw(4,4)=stdW^2;  % Size of Rw = Mord by Mord = Nx by
Nx

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% CONSTRUCT THE MEASUREMENT NOISE MATRIX Rv

    % TMB's Rv for the AGV mobility problem example problem:
                    Rv = stdV^2*eye(Nz,Nz);         % Create a diagonal matrix for Rv
```

```
                                                % Size of Rv = Mord by Mord = Nz by Nz

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% INITIALIZE PLOT MATRICES
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

                    P = zeros(Nx,Nsamples);         % Corrected covariances
                    P(:,1) = diag(Pc);              % Initial covariance - use the
diagonal

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% MAIN LOOP FOR BUILDING THE TRUE COVARIANCE MATRIX
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

            for k = 2:Nsamples

                    %-------------------------------------------------------------------
                    % For the AGV Node mobility problem:
                    %-------------------------------------------------------------------
                    % DEFINE VARIABLES TO BE PASSED TO THE EKF:
                    %-------------------------------------------------------------------

                            zk = z(:,k-1);          % zk is an Nz by 1 vector of
measurements to be passed
                                                    % to the EKF for the kth iteration

                            uk = u(:,k);            % uk is an Nu by 1 vector of inputs
to be passed to the EKF
                                                    % for the kth iteration

                            xk = x(:,k-1);          % xk is used for performance
evaluation only


                    %-------------------------------------------------------------------
                    % RUN THE EXTENDED KALMAN FILTER code EKF_TMB to produce results at
time t:
                    %-------------------------------------------------------------------

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PREDICTION:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            Xp = a*Xc + bu*uk;                  % (TMB's) state prediction for the linear
problem
            if abs(Xp(2)) > vmax                % Xp = Xhat(t|t-1) with velocity
limitation
                Xp(2) = sign(Xp(2))*vmax;
            end
            if abs(Xp(5)) > vmax
                Xp(5) = sign(Xp(5))*vmax;
            end

        %-----------------------------------------------------------------------------

            [HH] = Build_HH_TMB(xk);            % (TMB's) Jacobian

        %-----------------------------------------------------------------------------

            [hk] = Build_hk_TMB(Xp);            % (TMB's) measurement prediction hk =
h[xhat(t|t-1)]

        %-----------------------------------------------------------------------------

            Pp= a*Pc*a' + Rw;                   % (TMB's) state prediction covariance Pp =
Ptilda(t|t-1) for
                                                % the linear "stuctures" problem

        %-----------------------------------------------------------------------------
```

```
            zp = hk;                                % TMB's predicted measurement for the
nonlinear problem
                                                    % zp = zhat(t|t-1)

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% INNOVATION:
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

            inov = zk - zp;                     % (TMB's) innovation

       %-------------------------------------------------------------------------------
------------

            Rinov = HH*Pp*HH' + Rv;             % (TMB's) innovation covariance for
nonlinear problem
                                                % Note: HH is the Jacobian for measurement
equation

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% GAIN:
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

            K = (Pp*HH')*inv(Rinov);                % TMB's Kalman gain for the nonlinear
problem
                                                    % Note: HH is the Jacobian for the
measurement equation

%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
% CORRECTION:
%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

       %-------------------------------------------------------------------------------

            Xc = Xp + K*inov;                   % TMB's corrected state estimate for the
linear problem
            if abs(Xc(2)) > vmax                % with velocity limitation
               Xc(2) = sign(Xc(2))*vmax;
           end
           if abs(Xc(5)) > vmax
               Xc(5) = sign(Xc(5))*vmax;
           end

       %-------------------------------------------------------------------------------

            Pc = (eye(Nx)- K*HH)*Pp*(eye(Nx)- K*HH)' + K*Rv*K';   % (TMB's) corrected
covariance estimate for the
                                                                  % nonlinear problem.
Note: HH is the Jacobian
                                                                  % for the
measurement equation


%::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

                  %-------------------------------------------------------------------
                  %   SAVE RESULTS FOR PLOTTING, etc.
                  %-------------------------------------------------------------------

                                      P(:,k) = diag(Pc);       % Corrected covariance -
use the diagonal

            end
```

Q. BUILD EKF MEASUREMENT FUNCTION

This appendix presents the function that builds the mobile node measurement predictions within the EKF.

```
        function [hk] = Build_hk_TMB(Xp)
%
%
%         FUNCTION: Build_hk_TMB.m
%
%         PURPOSE: Function for building the Mobile node measurement prediction
%                  for the AGV mobile node problem.
%
%         SOURCE:                       Matlab M-file
%         VERSION:                      1.0
%         ORIGINATION DATE:             November 28, 2012
%         DATE OF LAST MODIFICATION:    November 28, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
%               % Xp = state prediction vector (Nx x 1)
%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
%               % hk = measurment prediction
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  EKF_Caller_TMB.m              % Markov Chain input builder code

%         CODES CALLED BY THIS FUNCTION:
%            1.  Sim_Parameters.m              % Passes simulation parameters

%         VARIABLES USED IN THE CODE:
%            1.  j                             % Used as increment for loop building hk
vector
%            2.  BS                            % matrix of base station coordinates in x
and y
%            3.  NBS                           % total number of base stations
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Define simulation parameters

            [Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax]=Sim_Parameters;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % Initialize variables
            hk=[];


 % BUILD THE MEASUREMENT PREDICTION VECTOR for the AGV Node

            for j = 1:NBS                                       % Measurement equation
(Nonlinear)

                  dk = sqrt((Xp(1)-BS(j,1))^2+(Xp(4)-BS(j,2))^2);
                  hk = [hk,z0(j) - 10*eta*log10(dk)];

            end

% Structure vectors for EKF
```

```
        hk=hk.';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

R. BUILD EKF JACOBIAN FUNCTION

This appendix presents the function that builds the Jacobian for the EKF for the mobile node problem.

```
        function [HH] = Build_HH_TMB(Xp)
%
%
%         FUNCTION: Build_HH_TMB.m
%
%         PURPOSE: Function for building the Jacobian for the AGV mobile node problem.
%
%         SOURCE:                        Matlab M-file
%         VERSION:                       1.0
%         ORIGINATION DATE:              November 28, 2012
%         DATE OF LAST MODIFICATION:     November 28, 2012
%
%         AUTHOR:       Timothy M. Beach (TMB)
%
%         INPUTS:
%            The user must specify the following inputs:
%               %
%
%         OUTPUTS:
%            The function produces the following results that are passed to the calling
program
%               % HH = Jacobian
%
%         CODES THAT CALL THIS FUNCTION:
%            1.  EKF_TMB.m                     % Code for EKF

%         CODES CALLED BY THIS FUNCTION:
%            1.  Sim_Parameters.m              % Passes simulation parameters

%         VARIABLES USED IN THE CODE:
%            1.  BS                            % matrix of base station coordinates in x
and y
%            2.  NBS                           % total number of base stations
%            3.  Xp                            % predicted state for current EKF
iteration
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Define simulation parameters

        [Mord,BS,NBS,x0,Ts,t,Nsamples,alpha,eta,z0,stdW,stdV,vmax]=Sim_Parameters;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 % CONSTRUCT THE JACOBIAN (HH) Necessary for the EKF:

        if NBS <3
            display('Need more Base Stations for triangulation. (Sim_Parameters.m)');
            return;
        end

        if NBS > 1
```

```
                HH = [-10*eta*(Xp(1)-BS(1,1))/(log(10)*((Xp(1)-BS(1,1))^2+(Xp(4)-
BS(1,2))^2)),0,0,-10*eta*(Xp(4)-BS(1,2))/(log(10)*((Xp(1)-BS(1,1))^2+(Xp(4)-
BS(1,2))^2)),0,0;
                      -10*eta*(Xp(1)-BS(2,1))/(log(10)*((Xp(1)-BS(2,1))^2+(Xp(4)-
BS(2,2))^2)),0,0,-10*eta*(Xp(4)-BS(2,2))/(log(10)*((Xp(1)-BS(2,1))^2+(Xp(4)-
BS(2,2))^2)),0,0;
                      -10*eta*(Xp(1)-BS(3,1))/(log(10)*((Xp(1)-BS(3,1))^2+(Xp(4)-
BS(3,2))^2)),0,0,-10*eta*(Xp(4)-BS(3,2))/(log(10)*((Xp(1)-BS(3,1))^2+(Xp(4)-
BS(3,2))^2)),0,0];
            end

            if NBS > 3
                HH = [HH;-10*eta*(Xp(1)-BS(4,1))/(log(10)*((Xp(1)-BS(4,1))^2+(Xp(4)-
BS(4,2))^2)),0,0,-10*eta*(Xp(4)-BS(4,2))/(log(10)*((Xp(1)-BS(4,1))^2+(Xp(4)-
BS(4,2))^2)),0,0];
            end

            if NBS > 4
                HH = [HH;-10*eta*(Xp(1)-BS(5,1))/(log(10)*((Xp(1)-BS(5,1))^2+(Xp(4)-
BS(5,2))^2)),0,0,-10*eta*(Xp(4)-BS(5,2))/(log(10)*((Xp(1)-BS(5,1))^2+(Xp(4)-
BS(5,2))^2)),0,0];
            end

            if NBS > 5
                HH = [HH;-10*eta*(Xp(1)-BS(6,1))/(log(10)*((Xp(1)-BS(6,1))^2+(Xp(4)-
BS(6,2))^2)),0,0,-10*eta*(Xp(4)-BS(6,2))/(log(10)*((Xp(1)-BS(6,1))^2+(Xp(4)-
BS(6,2))^2)),0,0];
            end

            if NBS > 6
                HH = [HH;-10*eta*(Xp(1)-BS(7,1))/(log(10)*((Xp(1)-BS(7,1))^2+(Xp(4)-
BS(7,2))^2)),0,0,-10*eta*(Xp(4)-BS(7,2))/(log(10)*((Xp(1)-BS(7,1))^2+(Xp(4)-
BS(7,2))^2)),0,0];
            end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%  END of M-FILE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. O'Rourke, Unmanned Vehicles for U.S. Naval Forces: Background and Issues for Congress, CRS Report for Congress, October 2006. Available: http:www.fas.org/sgp/crs/weapons/RS21294.pdf.

[2] E. Kuiper and S. Nadjm-Tehrani, "Geographical Routing with Location Service in Intermittently Connected MANETs," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 2, pp. 592–604, February 2011.

[3] N. Lechevin, C. Rabbath, and M. Lauzon, "A Decision Policy for the Routing and Munitions Management of Multiformations of Unmanned Combat Vehicles in Adversarial Urban Environments," *IEEE Transaction on Control Systems Technology*, vol. 17, no. 3, pp. 505–519, May 2009.

[4] J. Burbank, P. Chimento, B. Haberman, and W. Kasch, "Key Challenges of Military Tactical Networking and the Elusive Promise of MANET Technology," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 39–45, November 2006.

[5] S. Chakrabarti and A. Mishra, "QoS Issues in Ad Hoc Wireless Networks," *IEEE Communications Magazine*, vol. 39, no. 2, pp. 142–148, February 2001.

[6] P. Thulasiraman and X. Shen, "Interference Aware Resource Allocation for Hybrid Hierarchical Wireless Networks," *Computer Networks* (Elsevier), vol. 54, no. 13, pp. 2271–2280, March 2010.

[7] P. Thulasiraman, J. Chen, and X. Shen, "Multipath Routing and Max-Min Fair QoS Provisioning Under Interference Constraints in Wireless Multihop Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 716–727, May 2011.

[8] A. Vahdat and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," Tech. Report CS-2000–06, Department of Computer Science, Duke University, April 2000, Available: http://citeseerx.ist.psu.edu/viewdoc /download?doi=10.1.1.34.6151&rep=rep1&type=pdf.

[9] S. Jain, K. Fall, and R. Patra, "Routing in Delay Tolerant Networks," *Proc. ACM Special Interest Group on Data Communications (SIGCOMM)*, pp. 145–158, 2004.

[10] Z. Zaidi and B. Mark, "Mobility Tracking Based on Autoregressive Models," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 32–43, January 2011.

[11] E. Amar and S. Boumerdassi, “A Scalable Mobility-Adaptive Location Service with Kalman Based Prediction,” *Proc. IEEE Wireless Communications and Networking Conference (WCNC),* pp. 593–598, 2011.

[12] M. Musolesi and C. Mascolo, “CAR: Context Aware Adaptive Routing for Delay Tolerant Mobile Networks,” *IEEE Wireless Transactions on Mobile Computing*, vol. 8, no. 2, pp. 246–260, February 2009.

[13] Q. Yuan, I. Cardei and J. Wu, “An Efficient Prediction-based Routing in Disruption-Tolerant Networks,” *IEEE Transactions on Parallel and Distributed* Systems, vol. 23, no. 1, pp. 19–31, January 2012.

[14] H. Abu-Ghazaleh and A. Alfa, “Application of Mobility Prediction in Wireless Networks Using Markov Renewal Theory,” *IEEE Transactions on Vehicular Technology*, vol. 59, no. 2, pp. 788–802, February 2010.

[15] J. V. Candy, *Baysian Signal Processing, Classical, Modern, and Particle Filtering Methods*. Wiley: Hoboken, NJ, 2009.

[16] L. Mihaylova, D. Angelova, S. Honary, D. Bull, C. Canagarajah, B. Ristic, “Mobility Tracking in Cellular Networks Using Partical Filtering,” *IEEE Transactions on Wireless Communications*, vol. 6, no. 10, pp. 3589–3599, October 2007.

[17] Z. Zaidi and B. Mark, “A Mobility Tracking Model for Wireless Ad Hoc Networks,” *Proc. IEEE Wireless Communications and Networking Conference (WCNC),* pp. 1790–1795, 2003.

[18] J. V. Candy, *Model-Based Signal Processing*. IEEE Press and Wiley-Interscience, Hoboken, NH, 2006.

[19] A. Jazwinski, *Stochastic Processes and Filtering Theory*. Academic Press, New York, NY, 1970.

[20] Z. Yang and X. Wang, “Joint Mobility Tracking and Handoff in Cellular Networks via Sequential Monte Carlo Filtering,” *IEEE Transactions on Signal Processing*, vol. 51, no. 1, pp. 269–281, January 2003.

[21] Y. Takahashi, M. Rabins, D. Auslander, *Control and Dynamic Systems*. Addison-Wesley, 1970.

[22] P. Prasad and P. Agarwal, “Effect of Mobility Prediction on Resource Utilization in Wireless Networks,” *Proc. IEEE WCNC*, pp. 1–6, 2010.

[23] P. Prasad and P. Agarwal, “A Generic Framework for Mobility Prediction and Resource Utilization in Wireless Networks,” *Proc. IEEE COMSNETS*, pp. 1–10, 2010.

[24] P. Prasad, P. Agarwal, and K. Sivalingam, “Effects of Mobility in Hierarchical Mobile Ad Hoc Networks,” *Proc. IEEE CCNC*, pp. 1–5, 2009.

[25] A. Jardosh, E. Belding, K. Almeroth, and S. Suri, “Towards Realistic Mobility Models for Mobile Ad Hoc Networks,” *Proc. of ACM Mobicom*, pp. 217–229, 2003.

[26] C. Bettstetter, G. Resta and P. Santi, “The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks,” *IEEE Transactions on Mobile Computing*, vol. 2, no. 3, pp. 257–269, September 2003.

[27] G. Lin, G. Noubir and R. Rajaraman, “Mobility Models for Ad Hoc Network Simulation,” *Proc. IEEE INFOCOM*, pp. 1–10, 2004.

[28] R. Singer, “Estimating Optimal Tracking Filter Performance for Manned Maneuvering Targets,” *IEEE Transaction on Aerospace Electronic Systems*, vol. 6, no. 4, pp. 473–483, July 1970.

[29] T. Liu, P. Bahl, and I. Calamtac, “Mobility Modeling, Location Tracking, and Trajectory Prediction in ATM Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 6, pp. 922–926, August 1998.

[30] P. Eykhoff, *System Identification*. Wiley: Hoboken, NJ, 1974.

[31] G. A. Clark, “Angular and Liner Velocity Estimation for a Re-Entry Vehicle Using Six Distributed Accelerometers: Theory, Simulation and Feasibility,” Tech. Report, Lawrence Livermore National Laboratory UCRL-ID-153253, April 28, 2003.

[32] Y. Bar-Shalom, X. Li, and T. Kirubarajan, Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software. John Wiley & Sons, Inc., 2001.

[33] Y. Bar-Shalom, R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software*. Hobokoken, NJ: John Wiley & Sons, 2001.

[34] MATLAB Reference Manual. The Mathworks, Natick Massachussetts, 1993.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California